10-1-1997

The Design and modeling of input and output modules for an ATM network switch

Darin Murphy

THE DESIGN AND MODELING OF INPUT AND OUTPUT MODULES FOR AN ATM NETWORK SWITCH

by

Darin Murphy

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

Principle Advisor _____
Roy S. Czernikowski, Professor and Department Head

Committee Member _____
George A. Brown, Professor Emeritus

Committee Member _____
Muhammad E. Shaaban, Assistant Professor

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
October 1997

RELEASE PERMISSION FORM

Rochester Institute of Technology

The Design and Modeling of Input and Output Modules for an ATM Network Switch

I, Darin Murphy, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only.

_____

Darin J. Murphy

10/17/97

Date

ABSTRACT

The purpose of this thesis is to design, model, and simulate both an input and an output module for an ATM network switch. These devices are used to interface an ATM switch with the physical protocol that is transporting data along the actual transmission medium. The I/O modules have been designed specifically to interface with the Synchronous Optical Network (SONET) protocol. This thesis studies the ATM protocol and examines the issues involved with designing an ATM I/O module chipset. A model of the design was then implemented in both C++ and VHDL. These models were simulated in order to verify functionality and document performance.

The intent of this work is to provide the background and models necessary to aid in the further study and development of entire ATM switch architectures. The input and output modules are only two functional pieces of a complete ATM switch. The software models that have been implemented by this thesis can be integrated with the other necessary functional blocks to form a complete model of a working ATM switch. These functional blocks can then be rearranged and altered to assist in the study of how different switch architectures can effect overall network performance and efficiency. The input and output modules have been designed to be as flexible as possible in order to easily adapt to future modifications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

AAL, page 15
ATM Adaptation Layer. The highest layer in the ATM protocol stack. This layer is responsible for converting between higher layer PDU's and ATM cells.

ABR, page 14
Available Bit Rate. An ATM service category designed for connections that need low cell loss and high burst tolerance but can tolerate network delays and variation in transmission speed.

ATM, page 3
Asynchronous Transfer Mode. A high-speed connection-oriented switching technology that uses fixed length cells and can support multiple types of traffic. It is asynchronous in the sense that cells carrying user data need not be periodic.

B-ISDN, page 2
Broadband Integrated Services Digital Network. A high speed digital network standard that integrates voice, data, and other services. B-ISDN transmits at speeds above 1.544 Mbps and operates over ATM.

BER, page 4
Bit Error Rate. The rate at which bit errors occur in the physical transmission of a digital signal.

Broadband, page 2
Any service that provides transmission channels capable of supporting data rates greater than the ISDN primary rate.

Burst Tolerance, page 12
A quality of service parameter specifying the maximum length of time that a user can transfer data a peak cell rate over a particular connection.

CAC, page 39
Connection Admission Control. The portion of an ATM switch that is responsible for setting up all connections. The CAC must determine whether or not the network has enough resources to establish each connection that has been requested.

CBR, page 6
Constant Bit Rate. An ATM service category that provides a constant rate of data transmission. CBR is used for connections that require a guaranteed amount of bandwidth and low latency.

Circuit Switching, page 2
A method of data communications in which a dedicated path is established between two locations prior to the start of the communication process. Digital data is sent as a continuous stream of bits at a guaranteed bandwidth.

Connectionless, page 6
A type of data transfer in which information can be exchanged between locations without prior coordination.

Connection Oriented, page 6
A type of data transfer in which a logical connection is established between the communications endpoints.

CDV, page 12
Cell Delay Variation. A quality of service parameter specifying the change in interarrival times of each cell in a given connection.

CDVT, page 12
Cell Delay Variation Tolerance. The maximum allowed value for cell delay variation on a connection.

CLP, page 8
Cell Loss Priority. A 1-bit field in the header of an ATM cell that is used to determine which cells get discarded first when congestion occurs.

CLR, page 12
Cell Loss Ratio. A quality of service parameter that specifies the percentage of cells that a connection can lose in the network on an end-to-end basis. Given as the ratio of the number of lost cells to the number of transmitted cells.

CPCS, page 17
Common Part Convergence Sublayer. A sublayer of the convergence sublayer in the ATM protocol model. This layer performs functions that are common to all possible services.

CRC, page 40
Cyclic Redundancy Check. A mathematical algorithm used to detect and possibly correct bit errors after data transmission.

CS, page 17
Convergence Sublayer. A sublayer of the ATM adaptation layer in the ATM protocol model. This layer performs functions that are specific to a certain service.

CTD, page 12
Cell Transfer Delay. A quality of service parameter specifying the end-to-end time delay experienced by cells on a specific connection.

FDM, page 1
The division of a transmission facility into two or more channels by dividing the total available frequency band into several smaller bands, each of which is used as a separate channel.

Gbps, page 21
Giga-Bits Per Second. A unit used to measure the data transmission rate of a connection. A connection with a speed of 1 Gbps is transmitting 10^9 bits each second.

GFC, page 7
Generic Flow Control. The GFC is a 4-bit field found only in the header of a UNI cell. It is intended to be used in defining a simple multiplexing scheme.

HEC, page 8
Header Error Check. An 8-bit field in the header of an ATM cell that contains a code value used to detect and possibly correct errors in the 5-byte cell header.

IDN, page 1
Integrated Digital Network. The name given to the public switched telephone network after it was upgraded from analog to digital technology.

ISDN, page 2
Integrated Services Digital Network. A digital network standard that defines a set of services, capabilities, and interfaces supporting an integrated network and user interface.

Kbps, page 2
Kilo-Bits Per Second. A unit used to measure the data transmission rate of a connection. A connection with a speed of 1 Kbps is transmitting 10^3 bits each second.

LAN, page 4
Local Area Network.. A data and computer communications network confined to short geographic distances.

Mbps, page 2
Mega-Bits Per Second. A unit used to measure the data transmission rate of a connection. A connection with a speed of 1 Mbps is transmitting 10^6 bits each second.

MCR, page 12
Minimum Cell Rate. A quality of service parameter specifying the smallest cell transfer rate that a connection must always support.

Negative byte stuffing, page 29
A SONET network will insert negative stuff bytes into a frame when the transmission rate of the SPE is faster than the required frame rate. This practice preserves the alignment of the payload within the frame.

NNI, page 7
Network Node Interface. The interface between two ATM switches or two ATM networks.

OAM, page 4
Operations Administration and Maintenance. A system of network management functions that allow network administrators to troubleshoot and monitor network performance.

PCM, page 13
Pulse Code Modulation. A process in which a signal is sampled, and the magnitude of each sample with respect to a fixed reference is quantized and converted by coding to a digital signal.

PCR, page 12
Peak Cell Rate. A quality of service parameter specifying the maximum number of cells per second that a connection can transfer into the network.

PDU, page 6
Protocol Data Unit. A block of data exchanged between two entities via a protocol.

Positive byte stuffing, page 29
A SONET network will insert positive stuff bytes into a frame when the transmission rate of the SPE is slower than the required frame rate. This practice preserves the alignment of the payload within the frame.

PRS, page 20
Primary Reference Source. A precise master clock used to synchronize an entire network.

PSTN, page 1
Public Switched Telephone Network. A name referring to the public circuit switched network used to provide telephone service within the United States.

PT, page 8
Payload Type. A 2-bit field located in the header of an ATM cell that identifies the type of information contained in the data field.

PVC, page 6
Permanent Virtual Circuit. A virtual circuit within an ATM network that is maintained at all times, regardless of traffic flow. A permanent virtual circuit must be establish by a network administrator.

Packet Switching, page 1
A method of data communications in which messages are divided into units called packets. Each packet is then routed through the communications network independently.

SAR, page 17
Segmentation and Reassambly. A sublayer of the ATM adaptation layer in the ATM protocol model. This layer is responsible for both segmenting and reassembling data units and mapping them to and from fixed length cells.

SCR, page 12
Sustained Cell Rate. A quality of service parameter specifying the average number of cells per second that a connection can transfer into the network.

SDH, page 3
Synchronous Digital Hierarchy. The European equivalent of the SONET protocol.

SONET, page 3
Synchronous Optical Network. An international standard that defines a protocol for transmitting data at high speeds over a fiber optic network.

SPE, page 25
Synchronous Payload Envelope. The portion of a SONET frame that carries the user data. An SPE is allowed to float within a frame in order to account for the phase difference between the STS frame and the SPE.

SSCS, page 17
Service Specific Convergence Sublayer. A sublayer of the convergence sublayer in the ATM protocol model. This layer performs only service-dependent functions, if any exist.

STS-1, page 24
Synchronous Transport Signal Level 1. This is the most basic format of a SONET frame. It contains a total of 810 bytes and is transmitted once every 125 μsec.

STS-3c, page 37
The SONET frame format used to implement B-ISDN. A single STS-3c frame contains three STS-1 frames concatenated together. It transmits data at a rate of 155.52 Mbps.

SVC, page 6
Switched Virtual Circuit. A virtual circuit that is established and terminated upon request. When creating a switched virtual circuit, users must identify a destination address and all of the desired performance parameters.

TDM, page 1
Time Division Multiplexing. A data communications technique that assigns available bandwidth to users using predefined time slots. Connections take turns using the transmission channel. Time division multiplexing is the traditional method of sharing network resources.

UNI, page 7
User-to-Network Interface. A protocol which defines how ATM end users connect to private and public ATM networks.

VBR, page 6
Variable Bit Rate. An ATM service category that supports predictable data streams within bounds of average and peak traffic constraints. This service category is subdivided into both VBR Real-Time and VBR Non-Real-Time.

VCI, page 8
Virtual Channel Identifier. The VCI is a 16-bit field found in the header of ATM cells that identifies a particular virtual channel within a virtual path.

VPI, page 8
Virtual Path Identifier. The VPI is a field found in the header of ATM cells that is used to group virtual channels into paths for routing purposes.

WAN, page 4
Wide Area Network. A high speed data network used to connect communications equipment that is located a considerable physical distance apart.

1 Introduction

Throughout the history of the telecommunications industry, networks to handle specific types of information have been designed and built separately. For example, the public switched telephone network (PSTN) has been developed with the sole intention of handling analog voice communications. Several different data transmission networks and protocols have been designed for the purpose of computer communications. In addition, radio and television services are provided through the use of broadcast networks. Although each of these networks can perform its intended purpose effectively, they are generally not very efficient when trying to provide other forms of service. For instance, the modulation and demodulation process that is necessary when sending digital computer data over the telephone network requires extra interface hardware, is inefficient, and also limits the maximum transmission rate that can be achieved. The need for so many separate networks is based on the fact that different types of network traffic demand different performance in terms of bandwidth, end-to-end delays, and error rates.

The ideal telecommunications infrastructure would be a single network capable of efficiently handling all possible forms of communication. Digital switching and transmission technology is the first step in making such a network a reality. Digital transmission has many advantages over analog transmission which can greatly improve the performance and capabilities of telecommunications networks. Digital signals are less susceptible to noise, easier to regenerate, and easier to multiplex. [1, p. 5] In addition, digital technology enables networks to take advantage of the many benefits of packet switching technology. Packet switching is only possible on digital networks due to the fact that digital switches use time division multiplexing (TDM) to merge multiple low-speed connections onto a single high-speed line. Traditional analog switches use frequency division multiplexing (FDM) for the same purpose. The use of digital packet switching allows different types of communication traffic to be carried by the same network. The analog switching and transmission facilities originally used in the PSTN are rapidly being replaced by digital technology. This improved network is now being referred to as the integrated digital network (IDN).

1.1 Integrated Services Digital Network (ISDN)

Despite the use of faster and more reliable digital technology, the IDN is still a circuit switched network that is used primarily for voice communications. In order to achieve the goal of a single network that can provide a wide range of telecommunications services, the IDN must be expanded. This need was recognized in 1972 by the International Telecommunication Union-Telecommunication Standardization Sector (ITU-T). At that time, work began on a set of standards known as the integrated services digital network (ISDN). The first set of international standards for ISDN was not adopted until 1984. [1, p. 7] The objective of ISDN is to integrate user access to a variety of telecommunications services by defining a common set of interfaces. In addition to voice communications, ISDN incorporates a host of other non-voice services including data and video. ISDN uses end-to-end digital connectivity in order to provide all of these services on a single network.

The ISDN standards specify two types of user interfaces. The first provides a total bandwidth of 144 Kbps and is referred to as the basic rate. This user interface consists of two 64 Kbps channels and a single 16 Kbps channel. ISDN also offers a primary rate interface which consists of twenty-three 64 Kbps channels and a single 64 Kbps channel. The primary rate provides a total bandwidth of 1.536 Mbps. [1, p. 7]

The implementation of ISDN has been hindered by several factors. These include the slow completion of standards, insufficient end-user applications, incompatibility between different manufacturers' switches, and the inefficiency associated with supporting transmission rates slower than 64 Kbps. Furthermore, ISDN cannot provide the data rates necessary to implement new high-bandwidth services such as video teleconferencing. Although it is possible to combine multiple basic and primary rate channels to obtain increased bandwidth, this solution introduces another problem. Information transmitted over different channels may experience different delay times. [13, p. 6] Therefore, providing reliable real-time services using this method becomes much too difficult.

1.2 Broadband Integrated Services Digital Network (B-ISDN)

The need for greater bandwidth was quickly realized, and in 1985 the ITU-T began work on a new standard called broadband ISDN (B-ISDN). The main goal of B-ISDN is to expand ISDN to provide broadband services at rates of 150 Mbps and higher. The B-ISDN standard specifies the use of fiber optics as the physical transmission medium. Theoretical transmission rates of up to 10^{15} bps and very low error rates make fiber optics ideal for carrying all types of bandwidth-intensive telecommunications services. [1, p. 9]

1.2.1 Implementing B-ISDN using SONET

An important aspect of B-ISDN is the fact that the synchronous optical network (SONET) standard was chosen as the physical transmission protocol used to regulate the flow of digital information across the fiber optic medium. The SONET protocol was first developed by Bell Communications Research in 1985 to provide an optical transmission standard that would allow different manufacturer's equipment to be compatible. In 1988, the ITU-T adopted an international standard referred to as the synchronous digital hierarchy (SDH) that is mostly compatible with SONET. Presently, SONET is used in North America and SDH is used in Europe. [1, p. 9]

SONET was chosen as the physical layer protocol for B-ISDN for several reasons. First, SONET provides access to lower speed channels without the need to first demultiplex the entire high-speed signal. In addition, multiplexing and demultiplexing techniques using SONET are relatively simple. The SONET protocol also offers a large set of operations and maintenance capabilities, and it can be easily expanded to handle higher transmission rates in the future. Furthermore, adhering to the SONET standard allows different manufacturers to produce compatible networking equipment. [1, p. 10]

The use of SONET is also advantageous from a technical perspective. The SONET protocol uses a synchronous frame structure for a basic signal rate of 51.84 Mbps. Bytes in each frame can be interleaved to create a variety of different transmission rates and frame formats. Integer multiples of the basic signal are used to form higher bandwidth channels.

Networking operations and connection maintenance functions are conducted through the use of layered overhead and embedded data communications channels. SONET also provides a procedure for automatic protection switching. [1, p.10]

1.2.2 Implementing B-ISDN using ATM

One of the most significant advancements that the B-ISDN standard incorporated was a new switching and multiplexing technology referred to as asynchronous transfer mode (ATM). The ITU-T introduced ATM in 1988 in an attempt to provide the flexibility and high bandwidth that B-ISDN services require. ATM is a packet switched protocol in which information is partitioned into 53-byte cells and then asynchronously multiplexed for transport across the network. ATM networks provide high-throughput, low-delay, service-independent transport for all types of traffic, thus allowing different telecommunications services to be implemented over the same network. In addition to supplying integrated transport of all information types, ATM also supports dynamic bandwidth allocation and can make efficient use of network resources by means of statistical sharing. The ATM protocol can support all existing services and is flexible enough to adapt to other services that may be developed in the future.

Designers of the ATM protocol were able to take advantage of the high reliability of modern digital transmission techniques in order to greatly simplify ATM networks. For example, fiber optics have a very low bit error rate (BER) around the order of 10^{-12} as opposed to 10^{-6} for copper wire. [2, p. 1-3] Since this is the case, all error control and flow control functions are performed on an end-to-end basis rather than being performed within the subnet. This reduction in the amount of overhead associated with each packet allows ATM switches to process packets at high speeds and still provide reliable connections.

An important and unique feature of both the ATM and SONET protocols is that they can operate on both LANs and WANs. This allows for a seamless interface between these two types of networking environments. Figure 1 illustrates the interaction between ATM and SONET. In a B-ISDN network, SONET acts as the physical carrier system for transporting ATM cells. The SONET network provides a service to the ATM traffic. In

addition to the basic transport services, SONET provides operations, administration, and maintenance (OAM) functions. ATM provides direct services and interfaces to the user applications, as well as performing switching operations between SONET communications links.

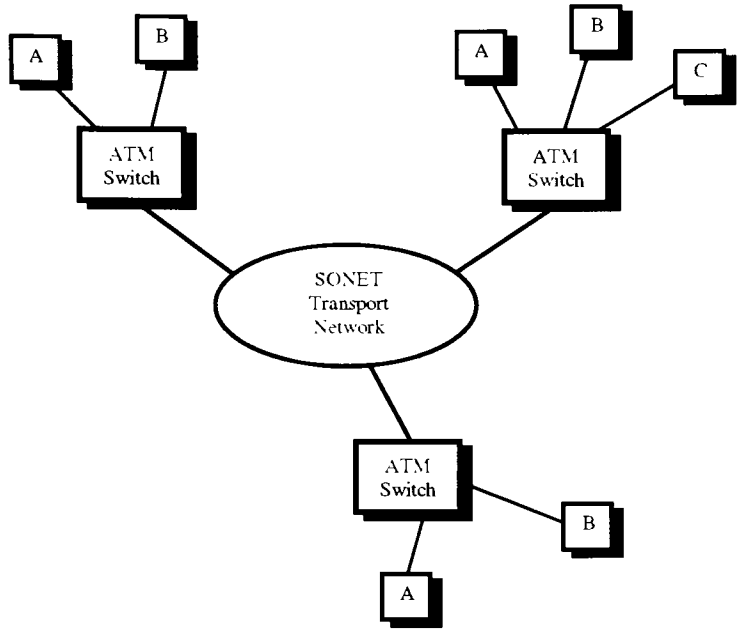


Figure 1 The interaction between SONET and ATM in a network [3, p. 19]

1.3 The interface between SONET and ATM

The important job of providing an interface between an ATM switch and the SONET transport network is the responsibility of the input and output modules contained within each ATM switch. Each input port of an ATM switch has an input module attached to it. The input module has several important functions. It must first be able to terminate the optical SONET signal and convert it to an electrical one. The overhead information associated with each SONET frame must be processed, and each valid ATM cell must be extracted from the frame. The input module is also responsible for ensuring that the header information within ATM cell has been received without error. Once this has been

accomplished, the input module will pass each cell to the switch fabric where it will be routed to the correct output port.

Each ATM switch also has an output module connected between each output port and the SONET transport network. The output module basically performs all of the same functions as the input module only in reverse. It must calculate an error checking code for each cell header and map each ATM cell into an outgoing SONET frame. The SONET overhead bytes must be generated, and the electrical signal used by the switch must be converted into an optical one prior to transmission over the fiber optic transport network.

The intent of this work is to provide the background information and models necessary to aid in the further study and development of entire ATM switch architectures. Clearly, the input and output modules are critical pieces of an ATM switch. Although input and output modules can be designed to interface with different transport protocols, those that interface with SONET are of particular interest due to their use in the implementation of B-ISDN. Therefore, a detailed study of input and output modules for ATM switches will be beneficial to the design and improvement of high speed digital networks.

2 The ATM Protocol

Asynchronous transfer mode is a connection-oriented switching and multiplexing technology designed to provide a wide range of services. Data is transferred using fixed length packets known as cells. An ATM network will take a user's digital data, segment it into multiple cells, and then multiplex these cells into a single bit stream which is transmitted across a physical medium. The ATM protocol is asynchronous in the sense that cells need not be transmitted periodically.

Each cell that is sent into the network contains addressing information that establishes a virtual connection between the source and the destination. All cells in a transmission are delivered in sequence over this same virtual connection. ATM provides either permanent virtual connections (PVC's) or switched virtual connections (SVC's). The use of virtual connections allows ATM to provide both connection oriented and connectionless services. Virtual connections can support either a constant bit rate (CBR) or a variable bit rate (VBR). ATM also supports multiple quality of service (QoS) classes in order provide separate performance parameters to different types of network traffic.

2.1 ATM Cells

The protocol data unit (PDU) of an ATM network is a cell. ATM standards define each cell to have a fixed-length of 53 bytes. Each cell contains a 5 byte header in addition to 48 bytes of user data. The 53 byte cell size was chosen as a compromise between network efficiency and packetization delay. Smaller cells make less efficient use of network resources because a greater percentage of each cell is overhead in the form of header information. However, smaller cells can be created and transmitted faster than larger cells. This reduces the delay time between the arrival of new cells on the receiver's end. Therefore, smaller cells are more favorable for the transmission of time critical information, such as voice communications. However, larger cells are better suited for services such as data traffic since less of the available bandwidth is used for overhead. The committees participating in

the development of the ATM standard agreed upon the 53 byte cell as an acceptable length for carrying varying types of traffic. [5, p. 205]

ATM defines two types of user interfaces. These are the user-network interface (UNI) and the network-node interface (NNI). A UNI is the interface between a user's equipment and an ATM public network service or into an ATM switch on a private enterprise network. An NNI is used in a connection between two ATM switches or two ATM networks. The format for an ATM cell is configured slightly differently for the UNI than for the NNI. Figure 2 shows the UNI format, while Figure 3 illustrates the NNI format.

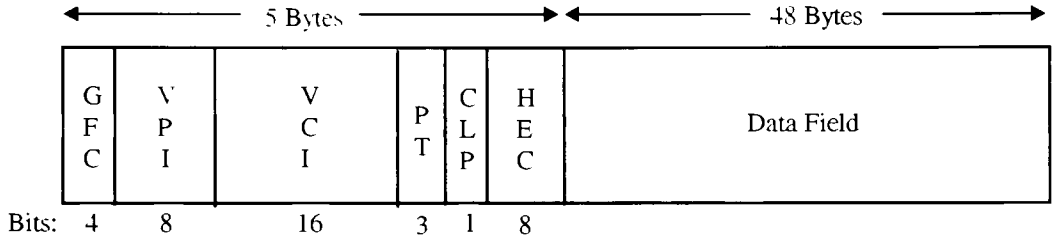


Figure 2 ATM cell format at UNI [1, p. 25]

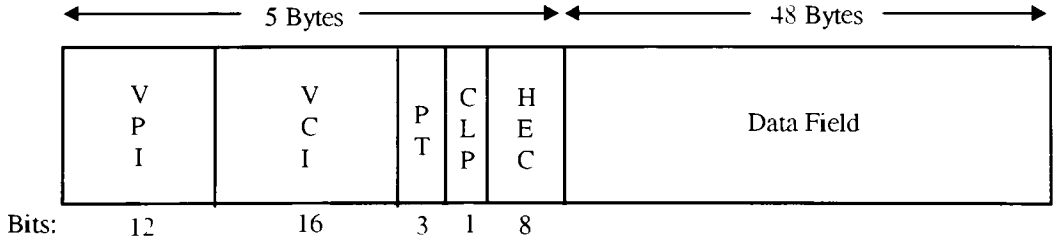


Figure 3 ATM cell format at NNI [1, p. 25]

Both the UNI and NNI cell formats contain a 5 byte header and a 48 byte data field. The main difference between the two formats is that the UNI cell contains a 4 bit generic flow control (GFC) field in the header. The GFC field is used to provide shared public network access when there is a single user access point servicing multiple terminal interfaces. [10, p. 426] The GFC ensures that each terminal will get equal access to the shared network bandwidth.

The payload type (PT) field in both the UNI and NNI cell formats uses 3 bits to distinguish the difference between cells that carry user data and those that carry maintenance traffic such as management and congestion information. Both header formats also contain a single bit referred to as the cell loss priority (CLP) field. This bit indicates whether a cell has high or low priority. These two levels of priority are used by ATM switches to determine which cells to discard during periods of network congestion. The header error control (HEC) field is a single byte used by the physical layer to correct single bit errors and detect multiple bit errors in the cell header. No error checking is performed on the payload data by the ATM network.

The virtual path identifier (VPI) and the virtual channel identifier (VCI) fields are used by the ATM network to establish a virtual connection from the source to the destination. The UNI cell format uses 8 bits for the VPI field while the NNI allocates 12 bits for the same field. Both header formats have VCI fields that are 16 bits in length.

2.2 Virtual Connections

In ATM networks, users communicate with each other through the use of virtual connections. A virtual connection defines a logical networking path between two endpoints in the network. All cells traveling between two such nodes are sequentially transmitted over this connection. Connections are virtual in the sense that they are defined in software or in the memory of the networking devices. Therefore, a direct physical path is not reserved between the source and destination when the connection is established. The use of virtual connections allows many connections to share the same physical resource. Each connection uses the resource when it has traffic to send. When a connection is idle, other connections are free to use the same circuit. Therefore, ATM makes efficient use of networking resources by allocating bandwidth dynamically and allowing multiple users to share the same resources.

The use of virtual connections is the reason why ATM is considered a connection-oriented technology. Although user data is split into numerous individual packets, the use of virtual

connections ensures that each packet will travel the same path from source to destination. This technique also guarantees that packets will arrive at the destination in the exact order in which they were sent. Therefore, reordering of packets before reassembly is not an issue in connection-oriented networks. The virtual connections in ATM are both bidirectional and full duplex. The bandwidth of a virtual connection can be specified uniquely for each direction.

2.2.1 Permanent and Switched Virtual Connections

An ATM network can contain both permanent virtual connections (PVC's) and switched virtual connections (SVC's). PVC's are predefined connections that are left in place all the time. If information is not being transmitted over a PVC it does not use any bandwidth on the network. However, each of the switches through which that PVC passes will need to permanently store routing information pertaining to that connection in its memory. The necessary connection information to handle PVC's must be manually loaded into the switching tables of an ATM network when it is first configured. PVC's are advantageous due to the fact that once the network has been configured to recognize a particular PVC, no setup time is needed before a transmission can be sent along that path.

In contrast, SVC's can be set up and torn down dynamically. In order to establish a SVC, signaling messages between the user and the network are used to exchange information about the type of connection required. Information pertaining to a specific SVC is removed from the switching tables once that connection has been terminated. Therefore, SVC's do not permanently occupy space in the memory of network switches.

2.2.2 Virtual Paths and Virtual Channels

As shown in Figure 4, each physical link contains several virtual paths. These virtual paths are in turn subdivided into virtual channels. On a specific physical link, each virtual connection is assigned a unique VP/VC pair. The VP/VC pair assigned to a virtual connection may be different on each physical link, therefore header address translation

must be performed at each network node. The address of the virtual connection associated with each cell is stored in the VPI and VCI fields of the cell's header.

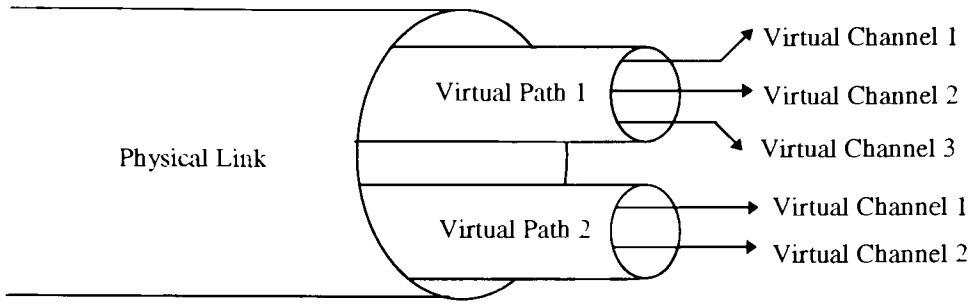


Figure 4 Relationship between physical link, virtual paths, and virtual channels [2, p. 2-7]

2.2.2.1 Routing through an ATM network

Address identifiers are unique to each physical link between two nodes in the network and are assigned locally at each switch. The necessary connectivity information is stored in tables located in the memory of each switch. Since a single connection is not assigned a unique address throughout the entire network, the addressing space required to keep track of all connections is reduced considerably.

When an ATM switch receives a new cell it first extracts the address identifier from the header. It then checks its switching tables in order to determine to which outgoing link the cell should be sent. Finally, the cell's address identifier is modified to contain the assigned local address of the outgoing link before the cell is placed into the appropriate output buffer.

An example of this routing technique is shown in Figure 5. Since VPI and VCI values must be unique on a specific transmission path, each switch maps an incoming VPI and VCI to the corresponding outgoing VPI and VCI based on the information stored in its routing tables. Table 1 shows the routing information necessary for each switch to perform address

translation on each incoming cell. This example is simplified by the fact that there is only a single link connecting each of the switches. In an actual network, the incoming link on which a cell was received would play a role in determining the new address identifiers. In addition, a switch would first have to decide onto which link to output a cell before performing the address translation.

In the example, the physical transmission path between Switch 1 and Switch 2 contains multiple virtual paths. At the ATM UNI of Switch 1, a virtual connection is established with a VPI of 1 and a VCI of 6. When a cell with this pair of address identifiers reaches Switch 1, the switch looks into its routing table and finds that an incoming VPI of 1 is mapped to an outgoing VPI of 12. Similarly, all incoming cells at Switch 1 with a VCI of 6 are given a VCI of 15 before being output on the appropriate link. This address translation occurs at each switch according to the routing information stored in the memory of that specific switch. Therefore, VPI's and VCI's are unique only on specific transmission paths between ATM switches and not throughout the entire network.

As the example in Figure 5 shows, the switch located at the destination UNI maps the VPI and VCI fields back to the values assigned by the source node. This ensures that the virtual connection has a consistent network address and that the address translation performed internally by the ATM network is transparent to all end-user applications.

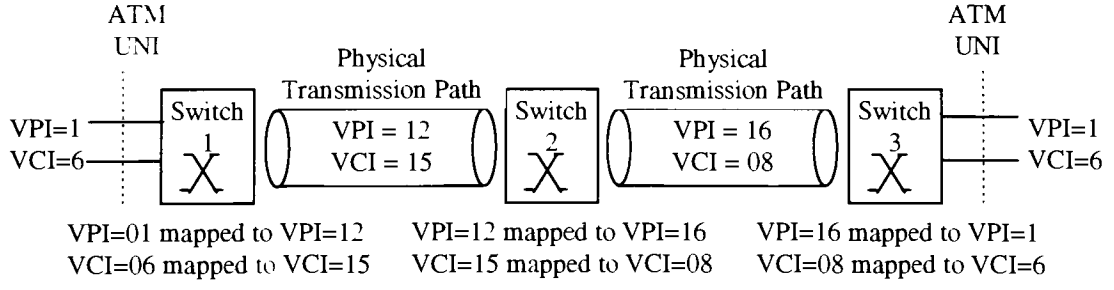


Figure 5 The use of VPI's and VCI's in forming a virtual connection [5, p. 211]

Switch Number	*Incoming VPI*	*Incoming VCI*	*Outgoing VPI*	*Outgoing VCI*
1	1	6	12	15
2	12	15	16	8
3	16	8	1	6

Table 1 Example of routing table information for network shown in Figure 5

2.3 Quality of Service

Each time a connection is established in an ATM network, a quality of service (QoS) must be specified for that connection. The quality of service assigned to a particular connection determines how the network handles the cells for that connection. This concept is necessary in order to enable ATM networks to carry different types of traffic effectively. The quality of service for a connection is defined on an end-to-end basis, thus ensuring the end user that particular performance parameters will be met.

2.3.1 Service Parameters

The ATM protocol defines seven quality parameters that are used when establishing connections. The first of these parameters is the peak cell rate (PCR). The PCR of a connection determines the maximum number of cells per second that the connection can transfer into the network. Although not necessary, the PCR is often set to be the maximum value possible for the given line rate.

Another QoS parameter is the cell delay variation (CDV). This parameter specifies the acceptable change in interarrival times of each cell. Variable delays in the cell stream can occur when cells from different connections are being multiplexed together. The cell delay variation tolerance (CDVT) represents the maximum allowable value for the CDV of a given connection.

A third QoS parameter, the sustained cell rate (SCR), determines the average number of cells per second that the connection is allowed to transfer into the network. A related parameter, burst tolerance, specifies the maximum length of time that the user can transfer information at the peak cell rate. If a connection sends traffic at peak cell rate for an entire burst tolerance period, that connection must then limit its cell transmission rate in order to meet the SCR requirement.

Three final quality of service parameters include the minimum cell rate (MCR), cell transfer delay (CTD), and cell loss ratio (CLR). The minimum cell rate is the smallest cell transfer rate that a connection must always support. The cell transfer delay is an end-to-end measurement of the time that it takes for a cell to travel from sender to receiver over a particular connection. The cell loss ratio is defined as the ratio of the number of cells lost by a connection to the number of cells transmitted over that same connection. This parameter is used to specify the maximum percentage of cells that a connection can lose in the network on an end-to-end basis.

2.3.2 Service Categories

The ATM protocol defines several service categories that are used to provide different levels of performance to different types of traffic. A service category is assigned to each virtual connection that is established in an ATM network. This category determines how the network prioritizes and allocates resources during a transmission over that specific connection. The amount of bandwidth allocated on each link and the buffer space allocated in each switch for a virtual connection will be determined in part by the service category that has been assigned to it. The ATM quality of service parameters are defined differently for each service class and are used to monitor the performance of each connection.

The service category given the highest priority in an ATM network is the constant bit rate (CBR) category. The CBR class is designed to support applications and connections that require a highly predictable transmission rate. CBR also ensures minimal delay and very low loss in the delivery of cells over a connection. This class is used to establish a connection that will supply a continuous stream of bits at a predefined constant rate. The CBR service

class is well suited for carrying real-time voice and video that uses pulse code modulation (PCM) to digitize the traffic stream. When establishing a CBR virtual connection, the PCR, CDVT, CTD, CDV, and CLR service parameters must be specified.

In addition to constant bit rate services, ATM can also handle variable bit rate (VBR) connections. Although the transmission rate of a VBR connection can vary, it must maintain an average bit rate over a specified period of time. This average rate of transmission is defined by the SCR (sustained cell rate) service parameter. The transmission rate can momentarily deviate from the SCR by bursting up to the maximum speed specified by the PCR (peak cell rate). VBR connections can be used to support applications that tend to transmit data in bursts. ATM defines two services categories that handle VBR traffic.

The "variable bit rate: real-time" (VBR-RT) service class is used to carry traffic that is fairly predictable but sensitive to delay and loss. VBR-RT connections are often bursty in nature, yet they require a strictly bounded delay. Packetized voice and video are examples of applications that require VRB-RT connections. Since many packetized voice implementations use compression and silence suppression techniques to allow statistical bandwidth gains, the transmission rate is often bursty. However, such applications still need to adhere to real-time operation timing constraints. The VBR-RT service category allows such connections to be handled by an ATM network.

The other variable bit rate service offered by ATM is "variable bit rate: non-real-time" (VBR-NRT). This service category differs from VBR-RT mainly in the fact that it is given a lower priority by the network. The VBR-NRT service class is designed to handle bursty connections that have less stringent delay requirements but still demand low cell loss. Applications that are well-suited for VBR-NRT are those that are tolerant of network delays and do not require a strict timing relationship to be maintained between the transmitter and the receiver. For instance, data applications that need high performance with low packet loss but do not need to be delivered in real time can communicate using a VBR-NRT connection. An example of this would be any information stored for later retrieval, such as and audio or video file.

Another service category provided by the ATM protocol is the available bit rate (ABR) class. An ABR connection provides high throughput for very bursty traffic while maintaining low cell loss rates. However, no timing or delay constraints are specified, therefore cell delay and delay variation are allowed to be high. For an ABR connection, the minimum amount of network resources are made available when needed. However, if part of the network bandwidth is idle, an ABR connection is allowed to use that excess capacity to send an extra burst of data. It will continue to transmit until it notices increased network delays or receives a congestion notification from the network. The ABR service class is designed to handle traffic that is not committed to any real-time delivery constraints. ABR is primarily used in the interconnection of LAN's.

The service category with the lowest network priority is the unspecified bit rate (UBR) class. UBR is used to establish connections that have absolutely no performance requirements. UBR connections are allowed to transmit only when adequate resources are available. If network congestion occurs, cells belonging to UBR connections are the first ones to be dropped. Since UBR connections have no objectives for cell delay or loss, no service parameters need to be specified. However, the PCR (peak cell rate) is generally set equal to the line rate. [2, p. 9-1] Table 2 summarizes the properties associated with each of ATM's service categories.

Service Category	Network Priority	Cell Delay and Delay Variation	Cell Loss	Burst tolerance
CBR	1	Low	Low	None
VBR-RT	2	Low	Medium	Low
VBR-NRT	3	High	Medium	Medium
ABR	4	High	Medium	High
UBR	5	High	High	High

Table 2 ATM service categories

2.4 ATM Protocol Structure

Similar to other communications systems, an ATM network can be modeled using a layered protocol stack. The ATM protocol structure consists of three distinct layers: the physical layer, the ATM layer, and the ATM adaptation layer (AAL). These three layers combine to provide end-to-end transport of a stream of information. Figure 6 illustrates each of these layers, the existence of any sublayers, and the corresponding responsibilities.

Layer Names		Functions
ATM Adaptation Layer	Convergence Sublayer	Service specific (SSCS) Common part (CPCS)
	Segmentation and Reassembly Sublayer	Segmentation and reassembly
ATM Layer		Generic flow control Cell header generation Cell VPI/VCI translation Cell multiplexing and demultiplexing
Physical Layer	Transmission Convergence Sublayer	Cell rate decoupling Cell delineation Transmission frame generation/recovery HEC field generation
	Physical Medium Dependent Sublayer	Bit timing Physical medium

Figure 6 ATM protocol stack [8, p. 20]

2.4.1 ATM Adaptation Layer

The ATM adaptation layer serves as an interface between the ATM layer and higher layer services. The AAL is responsible for converting ATM cells passed to it by the ATM layer into a format that can be recognized by the next highest protocol layer. This format is

usually referred to as a protocol data unit (PDU). Conversely, the AAL must also receive PDU's from higher layers and map them into ATM cells that are then passed on to the ATM layer.

Generally, adaptation only occurs at user network interfaces. These are the only points where higher layers will pass user data to the AAL. Upon receiving this data, the AAL will divide it up into cells and add the necessary header information. These cells are then passed to the ATM layer and transported across the network. Once they arrive at their final destination, the cells are passed back up to the AAL where the ATM cell overhead is removed. The data is then reassembled into its original PDU format and delivered to the next highest protocol layer.

Whenever the AAL receives a new cell, it performs a complete error check on the entire cell, including the payload data. If an error is detected, that cell is simply discarded. Error recovery is the responsibility of the higher layer protocols and is not performed by the ATM network. By checking for payload data errors only at the end user nodes, the ATM protocol is able to minimize delays at all nodes internal to the network. This type of error checking and recovery is not to be confused with the error checking that is performed on the header of each cell at every node throughout the network. This process uses the HEC field of a cell to validate the other four bytes of the header. This is necessary because the correctness of a cell's header information is essential to the proper delivery of that cell.

The AAL is further divided into two sublayers, the convergence sublayer (CS) and the segmentation and reassembly (SAR) sublayer. The CS performs a set of service specific functions that are necessary when interfacing between the ATM layer and higher layer protocols. The CS is comprised of its own two sublayers: the service specific convergence sublayer (SSCS) and the common part convergence sublayer (CPCS). The CPCS provides those functions which are required for all types of services. The SSCS provides functions that are necessary for specific services. This sublayer may not exist if an application does not require any service specific functions. [8, p. 44]

The SAR sublayer is the other sublayer of the CS. The SAR sublayer is responsible for both segmenting and reassembling user data so that it can be transmitted using fixed length ATM cells. At the transmitted side, the SAR sublayer takes the longer protocol data units (PDU's) received from the CS and segments them into the appropriate size to fit into the 48 byte cell payload area. In addition, any necessary headers or trailers are added. Once the cells have arrived at their destination, the SAR sublayer is responsible for reassembling the data into the format expected by the next higher level protocol layer.

2.4.1.1 AAL Service Classes

The AAL defines several different service classes that are used to differentiate between possible types of traffic that an ATM network may carry. These classes are separate from, but similar to, the quality of service categories used in relation to traffic management. The ATM protocol standard defines four main service classes which are designated Class A through Class D. Traffic for each of these classes needs to be handled differently by an ATM network. Therefore, there are also four AAL protocols defined to correspond with each one of the service classes. These protocols are referred to as AAL1, AAL2, AAL3/4, and AAL5.

2.4.1.1.1 Class A

The Class A service class supports constant bit rate, connection-oriented services which require a timing relationship between source and destination. This class is typically used to implement circuit emulation services which provide mapping of traditional TDM streams into ATM without any statistical gain. CBR voice and video are examples of applications that are typically implemented using Class A..

In order to support the features of Class A, the AAL1 protocol is defined to provide several specific functions in addition to the general AAL functions. These include the ability to use buffering to handle cell delay variations in order to provide a constant bit rate. In addition, AAL1 must recover the frequency of the source clock. Finally, AAL1 must detect bit errors

in the user data field and must also recognize and handle lost, discarded, duplicated, or misrouted cells. [2, p. 3-3]

2.4.1.1.2 Class B

Class B is similar to Class A in that it provides a connection oriented service in which a timing relationship exists between the sender and the receiver. However, Class B is designed to handle variable bit rate traffic rather than constant bit rate. Class B is intended to carry packetized video or similar applications.

The requirements for the AAL2 protocol associated with Class B tend to be quite complex due to the type of traffic it must handle. The AAL2 protocol must communicate variable bit rate information between end users, and must also maintain timing between source and destination. This protocol layer is also responsible for detecting errors or lost information that has not been recovered. A complete standard for AAL2 has not yet been defined.

2.4.1.1.3 Class C

Class C also provides a variable bit rate connection oriented service. However, Class C does not support a timing relationship between the transmitted and the receiver. Therefore, this class is well suited for handling connection oriented data traffic.

Class C is implemented using the AAL3/4 protocol. In addition to providing detection and signaling of errors in data, AAL3/4 also enables the multiplexing of multiple data streams over a single ATM virtual connection. AAL3/4 also supplies functions for carrying variable bit rate traffic in a manner similar to AAL2.

2.4.1.1.4 Class D

Class D is used to provide a connectionless communication service. It also supports a variable bit rate. A LAN interconnection is one example of an application that would be

well suited for Class D. LAN traffic is generally connectionless with a variable flow of information.

The AAL5 protocol provides Class D with the necessary functionality. AAL5 has the ability to segment and reassemble frames into cells. It can also detect errors in payload data. AAL5 is very similar to AAL3/4 except that it has less functionality and is therefore less complicated. [2, p. 3-3]

3 The SONET Protocol

The SONET protocol is a carrier transport technology for optical networks that utilizes synchronous operations between the network components. Although it is possible to use other physical layer protocols to carry ATM cells through a network, the B-ISDN specifications require the use of SONET. There are several features of the SONET protocol that make it a desirable transport method.

3.1 Synchronous Networks

When transmitting digital signals over a network, it is important for both the sender and the receiver to agree on the amount of time each bit is stable on the line. This allows the receiver to sample the signal at the proper times. When transmitting at relatively low bit rates, the sampling does not have to be as accurate because the signal does not change its value as often. However, as transmission rates increase, accurate sampling of the digital signal becomes increasingly important.

As more bits are transmitted each second, the amount of time that the receiver has to sample an individual bit decreases. Therefore, the clocks used by both the sender and the receiver to separate bit times must be synchronized to a high degree of accuracy. If the receiver's sampling clock is only slightly inaccurate, it might cause several bits to be missed or received incorrectly. This type of timing error is known as "slipping." Slipping occurs when both proper timing and detection of bits are lost.

All nodes in a synchronous network derive their clock from a single master clock, also referred to as the primary reference source (PRS). This scheme, known as a clock synchronization hierarchy, is illustrated in Figure 7. In such a clock hierarchy, timing is cascaded down from the master clock to each of the nodes in the network. For example, Node A will first synchronize its clock to the primary reference source. Node A will then use its own synchronized clock to control the frame rate for transmission to Node B. Upon receipt of a frame from Node A, Node B will use that frame to derive timing

information to synchronize its own clock. In turn, Node B uses its synchronized clock to send data to Node C. Node C is now able to use the frame that it received from Node B to extract timing information and synchronize its own clock. This same process is propagated through the network until timing information reaches all nodes and all clocks are synchronized. Updated timing information is transmitted each time a new frame is sent through the network.

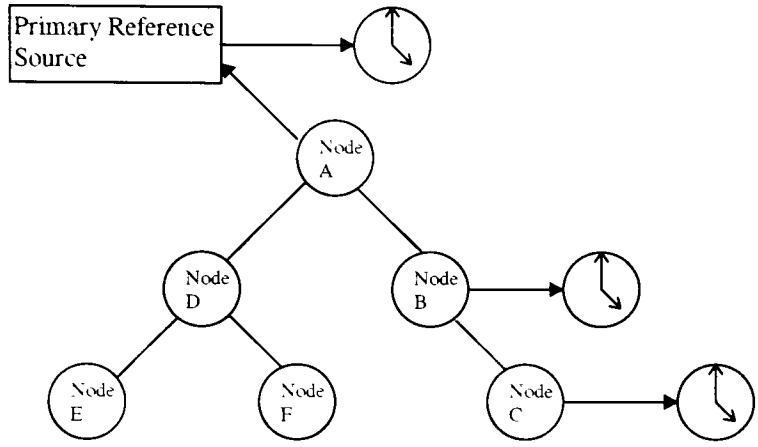


Figure 7 Clock synchronization hierarchy [4 , p. 110]

3.2 The Benefits of Optical Fiber

One of the most important aspects of the SONET protocol is that it was designed specifically to exploit many of the desirable features of optical fiber. Fiber optic cabling can easily obtain very high transmission rates of several Gbps. Furthermore, error rates for optic fibers are extremely low, even when transmitting at such high speeds. This is mainly due to the fact that the strength of a light signal is reduced only slightly after propagation through several miles of cable. Signal attenuation is a much greater concern in copper wire or coaxial cable. In addition, optical cables are advantageous from a security perspective. Information transmitted through the use of light is much harder to intercept than electrical signals.

Although more expensive than other transmission media, fiber optic cables have many favorable properties that make them easy to install and maintain. For instance, optical fiber

is extremely small considering the amount of bandwidth that it can provide. Therefore, fiber optic networks take up much less space and weigh considerably less than copper wire or coaxial networks with comparable performance. Since optical cables use light instead of electricity to transmit signals, they are not subject to any possible forms of electrical interference. In addition, fiber optics are flexible and are not affected by changes in temperature. These properties make fiber optic cabling ideal for installation in hostile environments.

3.3 Synchronous optical network (SONET)

SONET is an international digital transmission standard that takes advantage of both synchronous networks and optical fiber. As with other transport protocols, the purpose of SONET is to transport, multiplex, and switch digital signals that may contain many different types of user traffic. By establishing a single, international standard, SONET allows different manufacturers to produce compatible networking equipment. The SONET protocol has many important features that have allowed it to become widely used in international telecommunications networks.

The high speed and synchronous nature of the SONET protocol are two of its most attractive features. The SONET protocol uses a synchronous frame structure for a basic signal rate of 51.84 Mbps. Bytes in each frame can be interleaved to create a variety of different transmission rates and frame formats. Integer multiples of the basic signal are used to form higher bandwidth channels. The SONET protocol also offers a large set of operations and maintenance capabilities. These functions are provided through the use of layered overhead and embedded data communications channels. As the demand for more bandwidth in communications networks increases, the SONET protocol will receive more attention and use. The relationship between SONET and ATM is of particular importance because the B-ISDN standards define SONET as the physical transport protocol and ATM as the switching technology to be used in the implementation of this high speed network.

3.4 SONET multiplexing

One of the features of the SONET protocol that makes it so different from other transport protocols is its simple multiplexing scheme. Typically, other high speed communications networks have used a cascade of multiplexors to carry many connections over a single physical link. This configuration is shown in Figure 8.

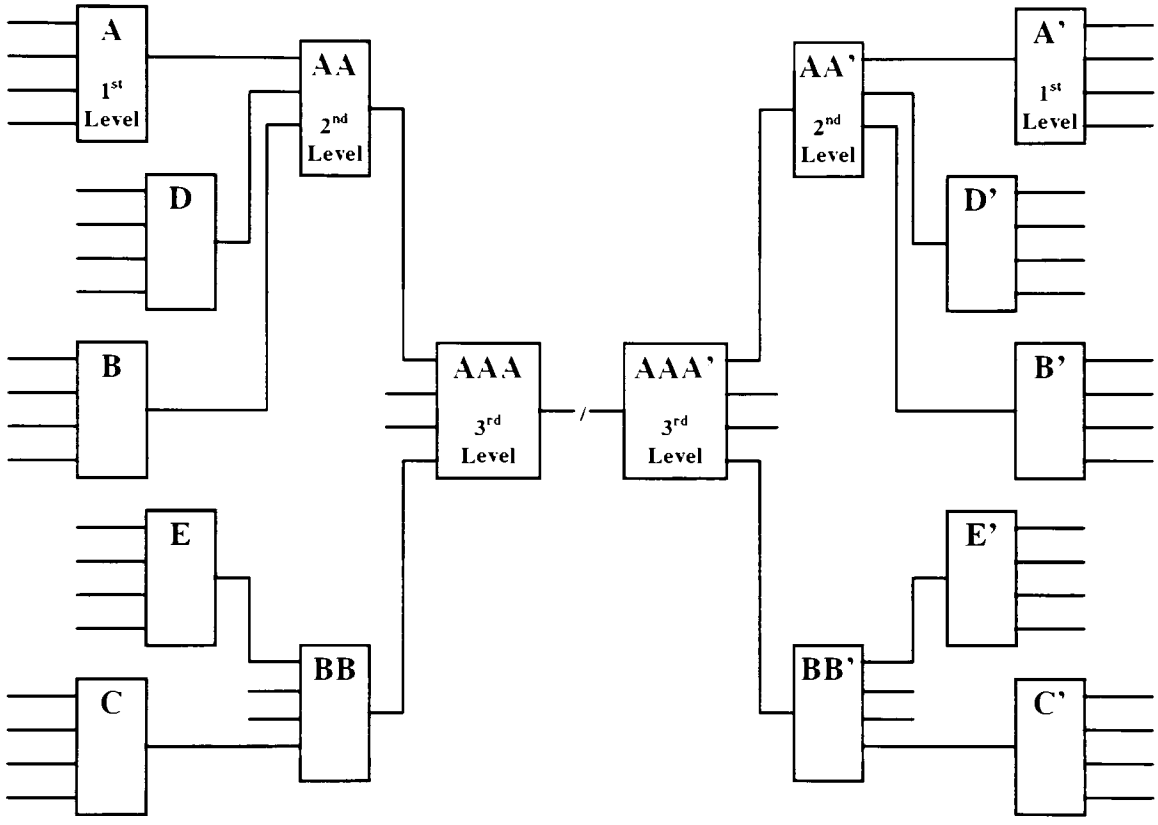


Figure 8 Traditional method of multiplexing across a high speed link [4, p.193]

With this configuration, it was necessary to multiplex a large number of slow speed connections together before they could be transmitted over a high speed link. Of course, the faster the link, the more stages of multiplexors were required. At the receiving end, the signal had to pass through several stages of demultiplexors in order to access a single slower-speed connection. Therefore, the entire structure had to be demultiplexed and then remultiplexed just to access a single circuit. This made the technique highly costly and inefficient.

Another problem with this approach is the issue of compatibility between multiplexing equipment. In order to operate correctly, each multiplexor at the transmitting end required a compatible demultiplexor to be present at the same level on the receiving end. Since the internal structure of each multiplexor is proprietary, this generally required that both the multiplexor and demultiplexor be manufactured by the same company. This added an unnecessary degree of difficulty to the design and implementation of high speed communication networks.

SONET uses a single multiplexing scheme that eliminates many of these problems. The SONET protocol defines a standard method of internal operation that ensures equipment produced by different manufacturers will be compatible. Furthermore, since SONET is an international standard, it allows high speed networks to span across the globe without added complexity.

One of the most important features of SONET is that it provides access to lower speed channels without the need to first demultiplex the entire high-speed signal. The many levels of multiplexing and demultiplexing are accomplished in a single step. This makes SONET a much more efficient transmission technique. Furthermore, SONET supports several different data transmission rates, and can easily be adapted to provide higher speeds as they become available in the future.

3.5 SONET frame structure

SONET uses a fixed length structure referred to as a frame to carry data across a physical link. There are several different frame formats that are supported by the SONET protocol. The most basic of these frames, and that from which all others are constructed, is the synchronous transport signal level 1 (STS-1). An STS-1 frame contains a total of 810 bytes. A new STS-1 frame is transmitted once every 125 μsec. Therefore, the minimum speed at which SONET can operate is 51.84 Mbps (810 bytes/frame * 8000 frames/sec * 8 bits/byte = 51.84 megabits/sec). Each byte in the frame can be regarded as part of a 64

Kbps communications channel. The STS-1 frame format can be represented as a 9-row by 90-column matrix of bytes, as shown in Figure 9.

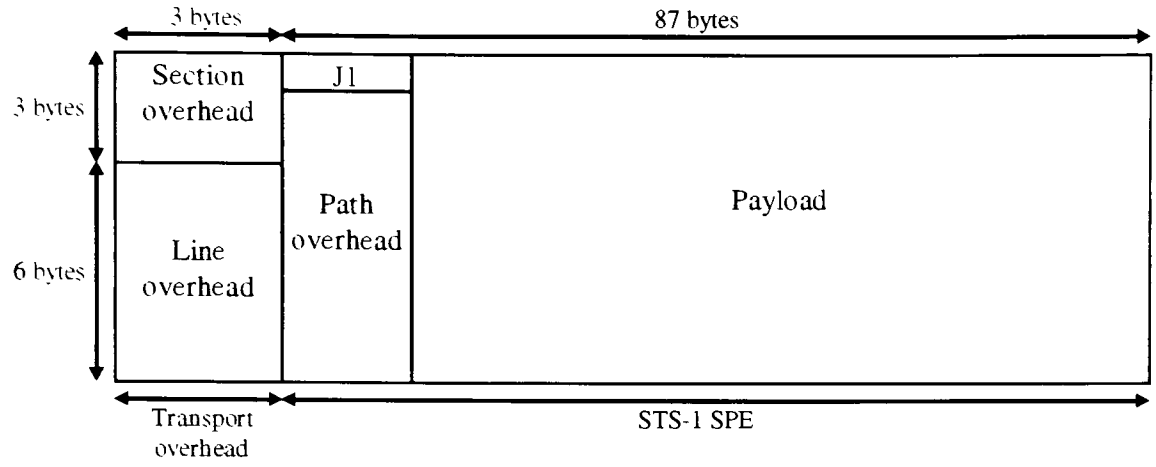


Figure 9 SONET STS-1 frame format [1, p. 103]

An STS-1 frame is transmitted row-by-row from left to right, starting at the top left of the frame and ending at the bottom right. The most significant bit of each byte is transmitted first. As illustrated in the Figure 9, each SONET frame consists of two main parts: the transport overhead and the synchronous payload envelope (SPE).

3.5.1 Transport overhead

The first three bytes of every row are reserved for transport overhead information used by administration and control functions. Each overhead byte is associated with a layer of the SONET protocol and is only processed by the network element that terminates that layer. This information can be further divided into section overhead and line overhead. The name and location of each overhead byte associated with a SONET frame is illustrated by Figure 10.

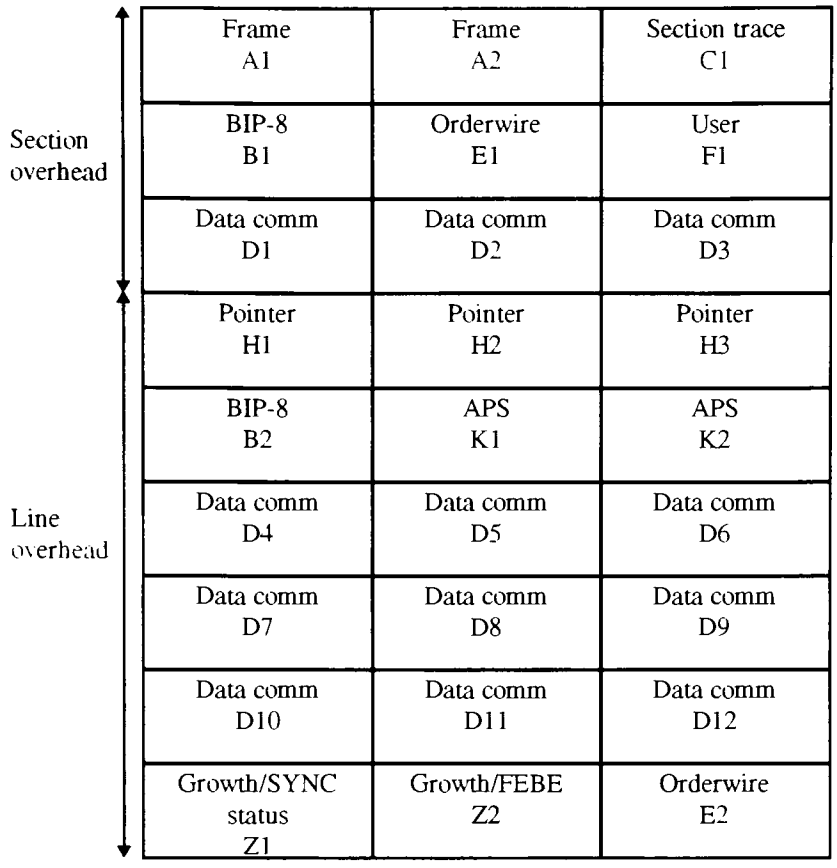


Figure 10 Transport overhead bytes [1, p. 104]

3.5.1.1 Section overhead

Each SONET STS-1 frame has nine bytes of section overhead. These bytes contain information necessary for such functions as frame alignment and section error monitoring. Framing is accomplished using the first two bytes, A1 and A2. Every STS-1 frame begins with the reserved bit patterns A1 = 11110110 and A2 = 00101000. The presence of these predefined bit patterns allow SONET network elements to check for proper frame alignment.

Other section overhead bytes include a BIP-8 byte, which is used for section error monitoring. An even parity checksum is calculated over all of the bits of the previous STS frame after it has been scrambled. This value is placed into the appropriate BIP-8 byte of the current STS-1 frame before it is scrambled. The orderwire section overhead byte acts as

a communications channel between regenerators, hubs, and remote terminals. Similarly, the section user byte provides a 64 Kbps channel for use by the network provider.

Finally, the section overhead of an STS-1 SONET frame contains three bytes that are reserved for a section data communications channel. These three bytes (D1, D2, and D3) together form a 192 Kbps communications channel. This channel is used by section terminating equipment to transmit alarm, maintenance, control, monitoring, and administration information. [1, p. 108]

3.5.1.2 Line overhead

Each SONET STS-1 frame also contains eighteen bytes of line overhead. The data stored in these bytes is used for such tasks as line maintenance, error monitoring, protection switching, and communication between line terminating equipment. The most important line overhead bytes are H1 and H2.

3.5.1.2.1 Payload pointer

The H1 and H2 bytes can serve three different purposes in an STS-1 frame. The first and most useful of these is to use H1 and H2 as a payload pointer. A payload pointer allows SONET to support the concept of a floating payload, which is an important and innovative feature of the SONET protocol. A synchronous payload envelope (SPE) is allowed to start anywhere within the physical SONET frame. A payload pointer is necessary to point to the start of the SPE. Allowing a payload to start somewhere in the middle of a frame requires that it also be allowed to span two consecutive frames.

The concept of a floating payload is a method of providing dynamic alignment of the payload within the STS frame. This allows SONET to compensate for phase differences between the SPE and the transport overhead, and in the frame rates as well. The payload pointer is created by combining the H1 and H2 bytes into a single 16-bit word. Figure 11

illustrates the significance of each bit when the H1 and H2 bytes are used as a payload pointer.

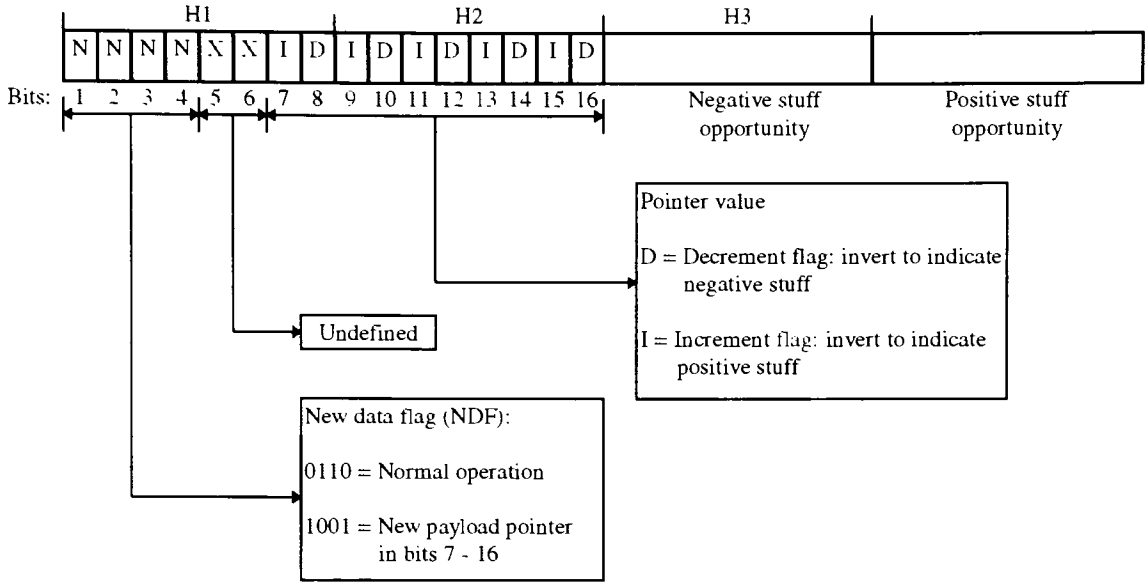


Figure 11 Payload pointer format in an STS-1 frame [1, p. 111]

The first four bits of the payload pointer are used to indicate when a new pointer value has been assigned. When this four bit new data flag (NDF) contains the bit pattern "1001", it is an indication that a change has occurred in the location of the STS payload, and that a new pointer value has been stored in bits 7 16. An NDF of "0110" signifies normal operation.

Following two undefined bits there are 10 bits (7 16) that are used to store the actual value of the payload pointer. A valid value for this pointer can be any decimal integer between the values of 0 and 782. This value represents the offset in bytes between the H3 byte of the line overhead and the first byte of the SPE within the STS-1 frame. This offset count does not include any of the transport overhead bytes. A pointer value of 0 indicates that the SPE begins with the byte immediately following the H3 byte.

If the SONET network encounters a timing difference in the SPE or transport overhead, it can adjust the pointer value to compensate. Whenever the transmission rate of the payload that is being placed into the STS-1 frame is less than the SONET frame rate, the SPE will

periodically slip back in time. This will cause the SPE to be one byte slower than the required frame rate. When this occurs, the payload indicator needs to be incremented by one. This is signified by inverting each of the I-bits in the H1 and H2 overhead bytes. When a frame containing inverted I-bits is encountered, the SONET network inserts a positive stuff byte into the frame. This stuff bytes contains no useful information and immediately follows the H3 line overhead byte. Its purpose is to allow the payload to slip back into the proper location within the STS-1 frame. All subsequent pointers will be updated to contain the new offset value. The process of positive byte stuffing is shown in Figure 12.

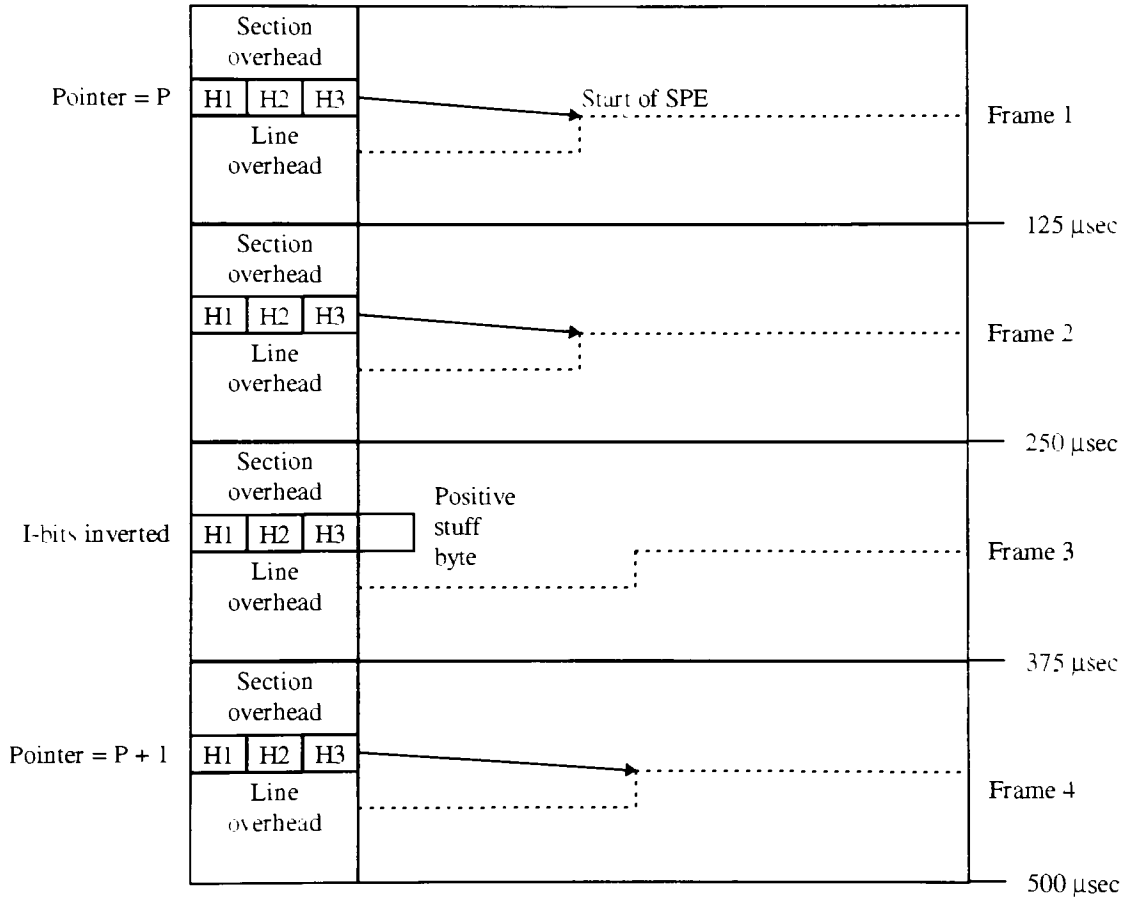


Figure 12 Positive byte stuffing [15, p. 79]

A SONET network is also capable of performing negative byte stuffing. This is necessary when the rate of the SPE is greater than the SONET frame rate. Such a scenario will cause the payload to periodically advance in time. The problem can be remedied by moving the

SPE forward one byte within the STS-1 frame. This is accomplished by first inverting all of the D-bits within bytes H1 and H2. The following frame will respond to the inverted D-bits by decrementing the pointer by one byte and updating all subsequent pointer values. The extra byte of information, called the negative stuff byte, is stored in the H3 byte of the transport overhead. Negative byte stuffing is simply a method of speeding up the SPE so that it will be properly aligned with the STS-1 frame. This process is demonstrated in Figure 13.

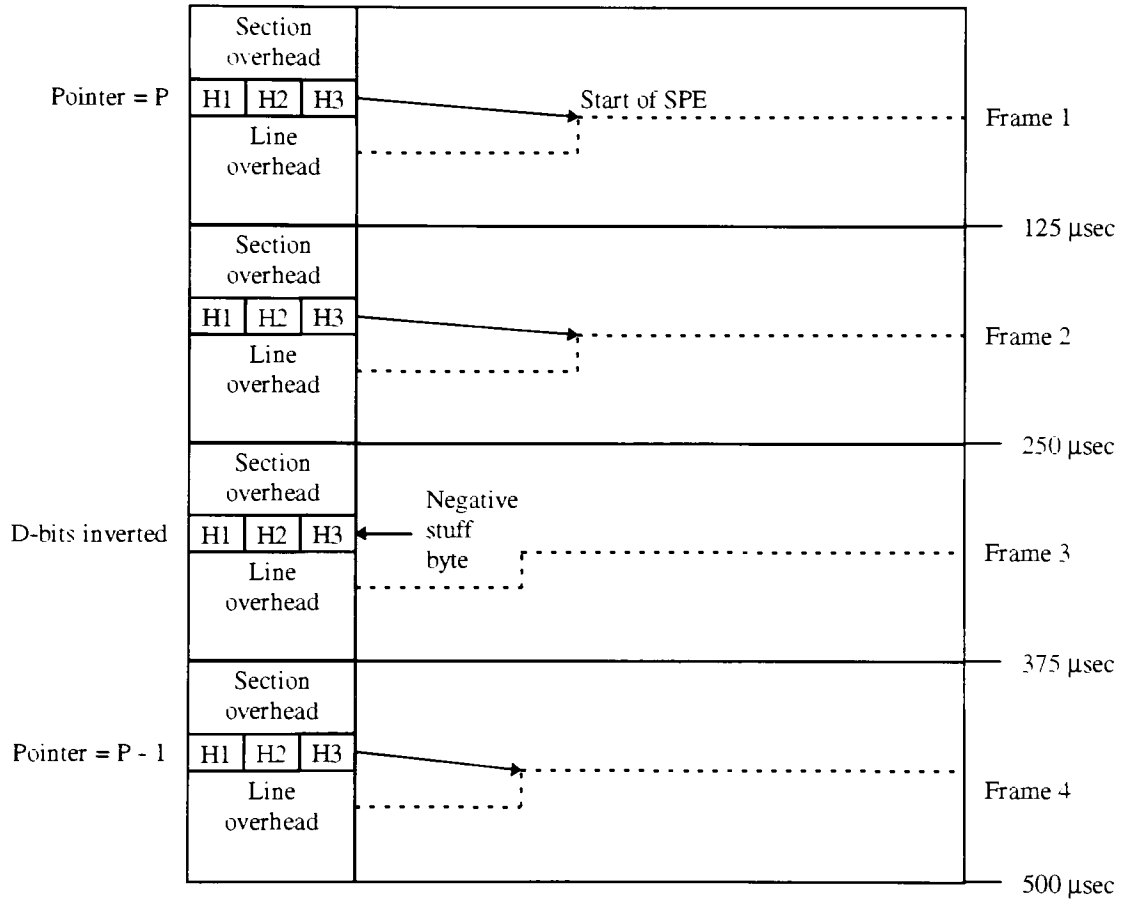


Figure 13 Negative byte stuffing [15, p. 79]

3.5.1.2.2 Other line overhead bytes

In addition to acting as a payload pointer, the H1 and H2 bytes of a SONET frame can serve two other purposes. These bytes can be used to indicate the concatenation of several

SONET STS-1 frames in order to form a higher speed signal. When this occurs, the values of H1 = "10010011" and H2 = "11111111" are assigned to all H1 and H2 bytes except those found in the very first STS-1 frame. The concept of frame concatenation to produce a faster data transmission rate is an important topic that is discussed in more detail later in Section 3.6.2.

The H1 and H2 bytes of a frame can also be used as a path alarm indication signal (AIS). A pattern of all ones in both H1 and H2 is used to inform path terminating equipment that a failure has been detected somewhere along the path.

Similar to the section overhead, the line overhead of a SONET frame also includes a byte (B2) for performing line error monitoring. An even-parity checksum is calculated using all of the line overhead and STS-1 payload bits of the previous frame before they are scrambled. This value is then placed into the BIP-8 byte of the current STS-1 frame before scrambling occurs.

The section overhead also includes several communications channels that can be used by separate pieces of line terminating equipment to exchange information. Two bytes, K1 and K2, are reserved to provide automatic protection switch (APS) signaling between line terminating equipment. There is also an express orderwire channel that uses byte E2 of the line overhead. Finally, a 576 Kbps data communications channel is defined using bytes D4 through D12. This channel is used to transmit messages regarding such functions as alarms, maintenance, control, monitoring and administration.

3.5.2 Synchronous payload envelope (SPE)

The synchronous payload envelope of a SONET STS-1 frame is the area in which the user data is stored. As discussed earlier, this payload area is allowed to float within the actual frame. The synchronous payload envelope of an STS-1 SONET frame contains 783 bytes.

3.5.2.1 Path overhead

Within the SPE, nine of the 783 available bytes are used for path overhead. A single path overhead byte is located in the first column of each row within the SPE. Figure 14 shows the location and purpose of each of these bytes.

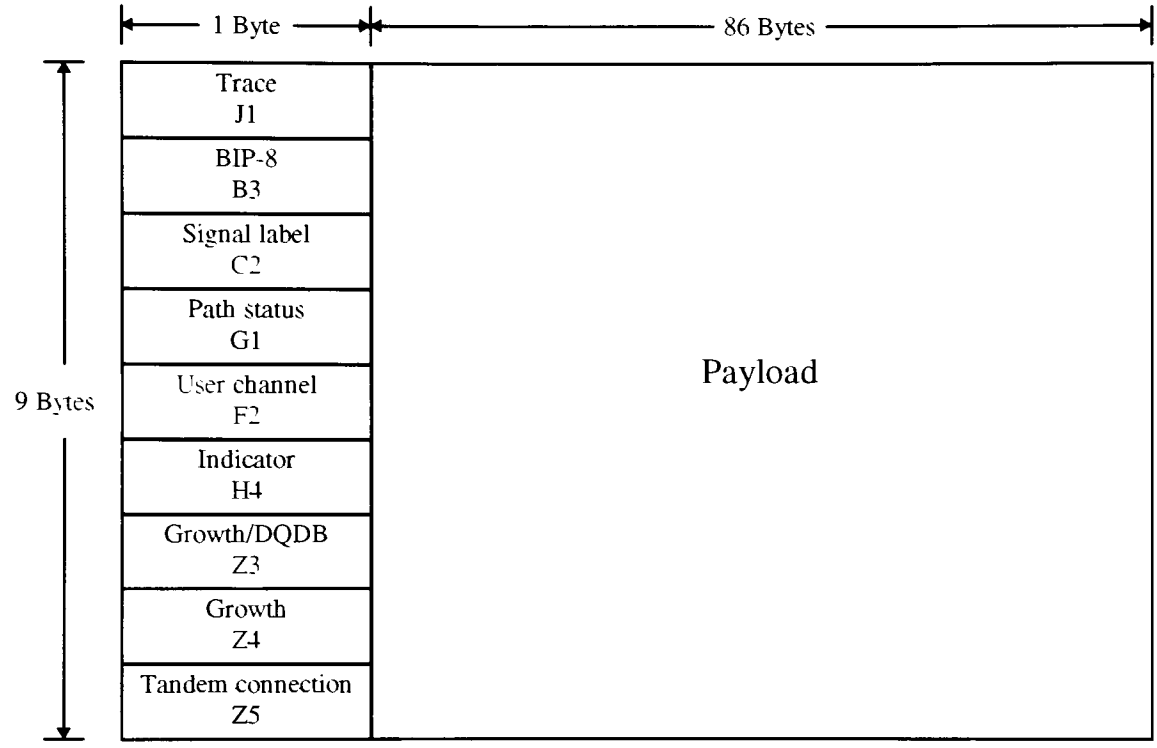


Figure 14 STS-1 SPE with path overhead bytes

Path overhead information is stored in the SPE itself because it is created at the same time as the SPE and is not accessed again until the SPE is terminated by a path terminating network element. The first byte of the SPE is the J1 path overhead byte. This byte provides an STS path trace mechanism. It is used to repetitively transmit a user selectable 64-byte string that enables path terminating equipment to verify that it connection with the intended transmitter is still open.

Similar to the transport overhead, the path overhead reserves a byte (B3) to perform BIP-8 error checking. The path overhead bytes also contain a path signal label (C2) and a path status (G1). The path signal label contains one of eight predefined codes that indicate the

formation of the STS SPE. The path status byte is used to relay information to the originating network element concerning the status and performance of a particular connection. In addition, there is a path user channel (F2) that can be used for communications by network providers. The remaining path overhead bytes are only loosely defined, and their purpose often depends on the specific application.

3.6 SONET signaling hierarchy

In addition to the basic 51.84 Mbps transmission rate of an STS-1 frame, the SONET protocol also provides higher bandwidth connections through the use of signal multiplexing and different frame formats. Each of the higher speed SONET signals is formed by multiplexing together several lower speed signals. These faster signals are designated as an STS-N frame, where N is an integer that represents the number of STS-1 frames that are contained in the new signal. Therefore, an STS-3 signal is three times faster than an STS-1 signal. Table 3 lists the different signal rates that are defined by the SONET protocol standards.

STS Level	Line Rate (Mbps)
STS-1	51.840
STS-3	155.520
STS-9	466.560
STS-12	622.080
STS-18	933.120
STS-24	1244.160
STS-36	1866.230
STS-48	2488.32
STS-96	4876.64
STS-192	9953.280

Table 3 SONET signaling hierarchy

Once the technology is available, multiplexing integrals greater than 192 can easily be incorporated into the SONET standards.

There are two separate techniques that are used to obtain the higher transmission rates defined by the SONET protocol standard. These are byte multiplexing and payload concatenation. The details of each will now be examined.

3.6.1 Multiplexing

The first method of forming high speed SONET signals is to byte interleave several STS-1 frames. If N STS-1 frames are multiplexed together, then the new signal will be of level STS-N. Regardless of the number of STS-1 frames that are combined, the new frame will still have a period of 125 μsec. Therefore, the new transmission rate of the signal will be N times the 51.84 Mbps rate of an STS-1 frame. This technique is illustrated by Figure 15. In this example, three STS-1 frames are passed through a single stage 3-to-1 multiplexer to create an STS-3 signal that has a line rate of 155.52 Mbps.

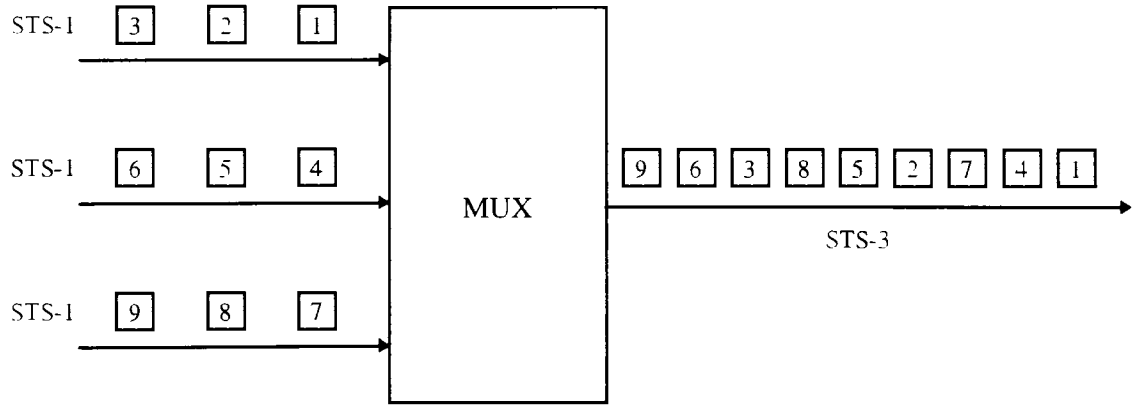


Figure 15 Formation of an STS-3 signal using single stage multiplexing

When an STS-N signal is created, the transport overhead bytes of the individual STS-1 frames need to be frame-aligned before they are interleaved. However, the SPE's do not need to be aligned because a separate payload pointer will exist to mark the beginning of each payload. Each individual SPE contains its own column within the path overhead

section of the byte. The general format of a frame created using this multiplexing technique is shown in Figure 16.

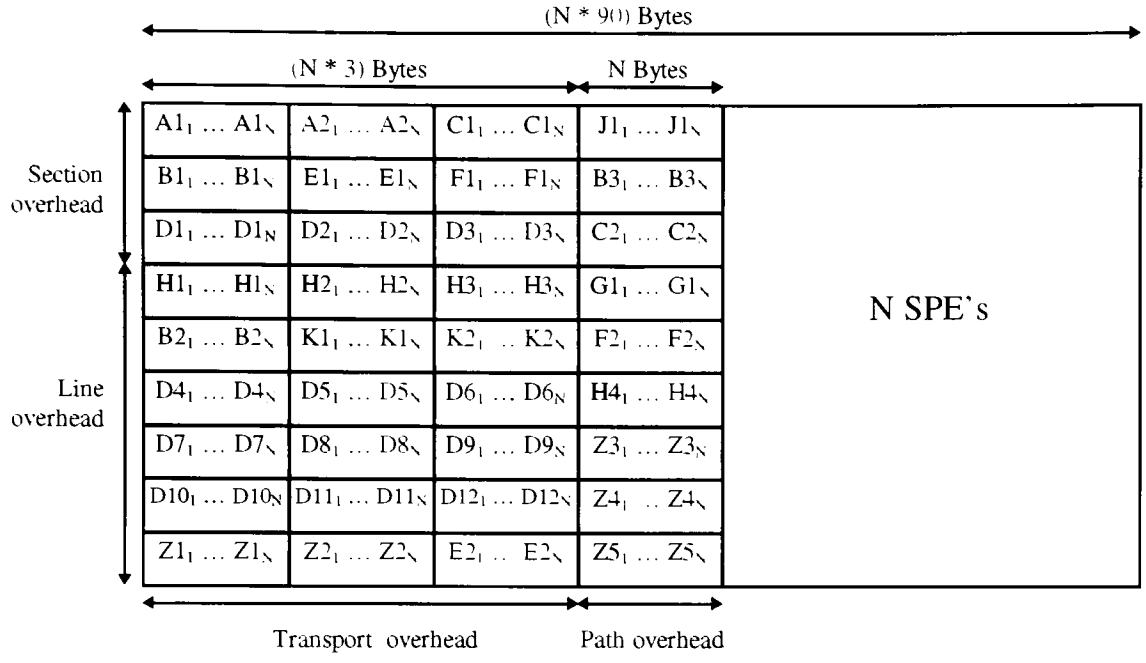


Figure 16 STS-N frame format

3.6.2 Concatenation

An alternative method for providing high speed SONET connections is referred to as concatenation. With this technique, multiple STS-1 frames and their payloads are phase aligned to form a single frame. This creates a single, larger synchronous payload envelope. Concatenated signals are referenced using the convention STS-Nc. The value of N indicates the number of STS-1 frames contained in the signal, and the 'c' denotes that the frames have been concatenated together to form the higher-speed signal.

Frame concatenation is advantageous in that the payloads of each STS-1 signal are combined into a single SPE that can be transported through the network as an autonomous entity. ATM standards specify an access rate of 155.52 Mbps for the implementation of the

B-ISDN UNI. This rate can be achieved using a SONET STS-3c signal. The frame format for such a signal is shown in Figure 17

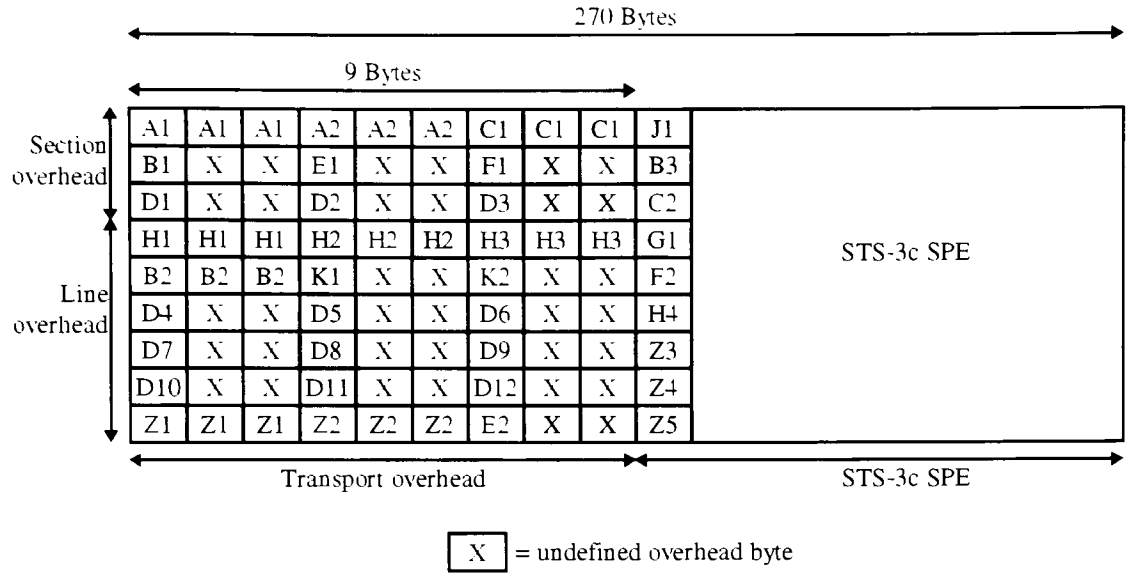


Figure 17 STS-3c frame format [1, p. 107]

With the previous method of multiplexing several STS-1 frames together to form a single high-speed signal, a separate set of path overhead bytes is required for each of the STS-1 SPE's. However, since a concatenated frame only contains a single SPE, it only requires one set of path overhead bytes. A concatenated frame still reserves all of the transport overhead bytes for each of the embedded STS-1 signals. However, these bytes are only defined for the first STS-1 within the signal. All subsequent transport overhead bytes are either undefined, or they are filled with predefined patterns to indicate that the frame contains a concatenated signal. Each concatenated frame still maintains a period of 125 μsec, thus allowing it to achieve the higher transmission rate desired.

4 ATM Switch Design

Although the main purpose of an ATM switch is to maintain virtual circuits by routing cells from input ports to output ports, there are many other tasks that must be performed in order for this to occur. In addition to handling user data, a switch must also communicate management and control information with other parts of the network so that critical services such as traffic control can be effectively implemented.

4.1 Components of an ATM Switch

A complete ATM switch can be modeled by grouping each of the necessary functions into several separate functional blocks. These blocks include input modules, output modules, switch fabric, connection admission control, and system management. The interconnections and flow of information between each of these functional blocks define the architecture of an ATM switch. The design of each of these blocks will depend on how the overall switch architecture is constructed. Each of the internal pieces of an ATM switch will now be examined in more detail.

4.1.1 Input Modules

The main purpose of the input module is to receive incoming cells and prepare them to be routed through the switch fabric. Therefore, a separate input module is necessary at each input port to provide an interface between the transport network and the ATM switch. Assuming that the SONET protocol is being used to transport ATM cells between switching nodes, each input module must first terminate the SONET signal and extract the stream of ATM cells. Several steps are necessary in order to perform this task. First, the optical signal used by the SONET network must be converted into an electrical signal that can be processed by the input module. The digital bitstream contained in this signal must then be recovered. Once the bitstream has been extracted properly, the SONET overhead can be processed and the task of cell delineation can begin. Cell delineation is necessary to determine the boundary between individual ATM cells. Finally, the input module must

perform cell rate decoupling, which consists of discarding any idle cells that were inserted at the transmitting end in order to adapt to the bandwidth capacity of the physical medium.

Once the SONET signal has been correctly terminated by the input module, only non-idle ATM cells will remain. These cells must next be prepared for routing through the switch fabric. In order to do so, error checking must be performed on each individual cell. The header error control (HEC) byte of the cell header is used to verify that the cell's header was received correctly. The payload of the cell is not checked for errors. Cells with single bit errors in the header are corrected, and those with multiple bit errors are discarded.

Depending on the design of the individual input module and the overall switch architecture, there are several other tasks that an input module may be required to perform. An input module may be designed to differentiate between cells containing user data and those carrying signaling or network management information. If a signaling cell is encountered, it must be properly routed to the connection admission control (CAC) functional block of the switch where it can be appropriately processed. Furthermore, if a cell containing network management information is discovered, it must be sent to the system management (SM) portion of the switch.

It is also possible for an input module to attach an internal tag to each cell that it handles. Such a tag can be used to keep track of information pertaining to performance and routing. The use of such a tag and the exact information that it contains depends strictly on the design of the switch. Since these tags are intended to be removed by the output module, they will not be passed between switches in an ATM network. They are only used to store and communicate information internally. The design of an input module will be discussed further in Chapter 5.

4.1.2 Output Modules

An output module is located at each output port of the switch fabric. The output module is similar to the input module, only it performs many of the same functions in reverse. Once

a cell has been routed through the switch fabric, it must be passed through an output module in order to be prepared for physical transmission over the transport network.

One of the main responsibilities of the output module is to generate the header error control (HEC) byte for each cell that it receives. This is accomplished by using a generator polynomial to calculate an eight bit cyclic redundancy check (CRC) code for the first four bytes of each cell header. This value is then placed into the fifth byte of the header before it is transmitted to the next node in the network. This HEC field is then used by the next node to check for errors in the header of that cell. The ATM network does not perform error checking on the user data, only on the overhead bits of each cell.

Another important function of an output module is to map the outgoing ATM cells into the payload field of a SONET frame. Since ATM is an asynchronous protocol while SONET is a synchronous one, it is necessary for each output module to perform cell rate decoupling by inserting idle cells into the SONET frame. These empty cells contain no useful information, but are necessary to fill unused time slots on the SONET network. Furthermore, the output module must generate the overhead information associated with each SONET frame that it produces.

An output module may be required to perform several other tasks, depending on the overall design of the ATM switch architecture. For example, if the input modules of the switch attach internal tags to each cell, then the corresponding output modules must be designed to remove each of these tags and process the information that they contain. The distribution and interconnection of functional blocks within the switch will also determine whether or not the output module will be responsible for combining signaling cells from the CAC and management cells from the SM with outgoing cells containing user data. Furthermore, it may also be necessary for an output module to perform a final translation of a cell's VPI and VCI values. Finally, the output module must convert the electrical signal used by the switch into an optical one that corresponds to SONET standards and can be transmitted over the next physical link. The design of an output module will be discussed further in Chapter 6.

4.1.3 Switch Fabric

The switch fabric is the core of an ATM switch. This functional block is responsible for transferring ATM cells between other functional blocks within the switch. This includes routing cells containing user data from the input modules to the appropriate output modules. Since the switch fabric is so important to the proper functionality and performance of an ATM switching system, there has been much research conducted in this one particular area of ATM switch design. It is beyond the scope of this discussion to go into detail concerning the many switch fabric architectures and performance issues associated with each. Only a general overview of the responsibilities of the switch fabric will be discussed here. For more information, refer to [14].

In addition to the correct routing of cells from input ports to output ports, the ATM switch fabric is responsible for buffering both incoming and outgoing cells. The location and types of buffers used are an important design issue for a switch fabric. Buffering is necessary for the case when more than one cell arrive at an input port simultaneously. Since the switch fabric can only route one cell at a time, any other cells at an input port must be buffered until they can be processed. In a similar manner, each output port can only transmit one cell at a time. Therefore, any cells that arrive while the output port is busy must be buffered until they are able to be transmitted.

Another issue related to buffering is that of congestion management. This concerns the manner in which a switch reacts if it is unable to handle all of the cells that it is receiving. If the switch's buffers are ever allowed to overflow, then cells will be lost. Therefore, it is often possible for the switch to assert backpressure to the sending node in order to slow the rate of transmission and minimize packet loss. However, if cell loss cannot be avoided due to heavy volumes of traffic, then the switch fabric must exercise a form of selective cell discarding based on loss priorities. Delay priorities associated with individual cells will also affect the order in which the switch fabric must schedule certain cells.

The switch fabric is also responsible for maintaining a certain level of fault tolerance. This function concerns the detection and analysis of failures within the network in order to

maintain reliable operation. The switch fabric must be able to detect faults and effectively recover from them. The desired degree of fault tolerance can be achieved by adding redundancy to the critical components of the switch fabric. This can be accomplished by either duplicating the entire fabric or adding redundancy within the fabric. In order to properly introduce and remove redundancies from the switch fabric, functional blocks designed to expand and concentrate cell traffic are necessary.

A switch fabric must also be able to handle multicasting. A multicast cell is one that has more than one output port as its destination. This can occur when information needs to be broadcast to several users. A cell may need to be routed to all output ports or only a few select ones. In either case, the original cell must be properly copied and delivered to the correct output ports.

These are only a few of the functions that a switch fabric may be responsible for. Of course, exactly which functions are implemented within a specific switch fabric will depend on the design of the overall switching system architecture and the specifications to which the switch is being designed. Since the switch fabric is such an important and complex piece of the complete ATM switch, a functional block diagram is often generated to demonstrate its design and implementation. A generic block diagram showing the basic functionality of a switch fabric is given in Figure 18.

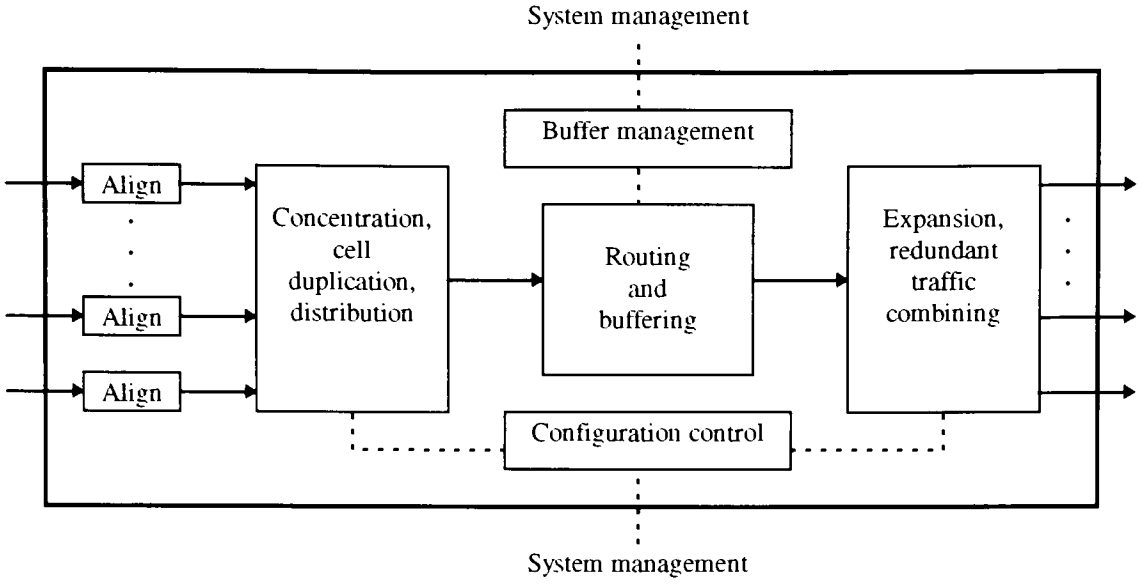


Figure 18 Functional block diagram of the switch fabric [1, p. 145]

4.1.4 Connection Admission Control

The connection admission control (CAC) is responsible for creating, maintaining, and terminating all virtual connections passing through the switch. When a user indicates that a new virtual connection is to be established, the CAC is responsible for negotiating certain QoS performance parameters that are to be guaranteed for the connection. These negotiations are based on the amount of available resources and the service contracts that already exist with other connections. If an acceptable agreement can be reached, the CAC will allocate the necessary resources for the new connection. These resources include the physical links that the connection is going to be carried over, and the amount of bandwidth that is to be reserved on each link. Throughout the life of that connection, it may be necessary for the CAC to renegotiate with the user due to network congestion or failure, or due to the demands of other connections to maintain a certain quality of service. The CAC can also decide to deny a requested connection.

Based on the network resources that are currently available, the CAC must decide whether or not it can provide the desired level of performance to the user. If the virtual connection is accepted, the CAC must then assign a VPI and a VCI to this new connection. The

switching tables must also be updated to contain the information pertaining to this new connection. This process will involve communicating with other functional portions of the ATM switch. A functional block diagram of the connection admission control module is shown in Figure 19.

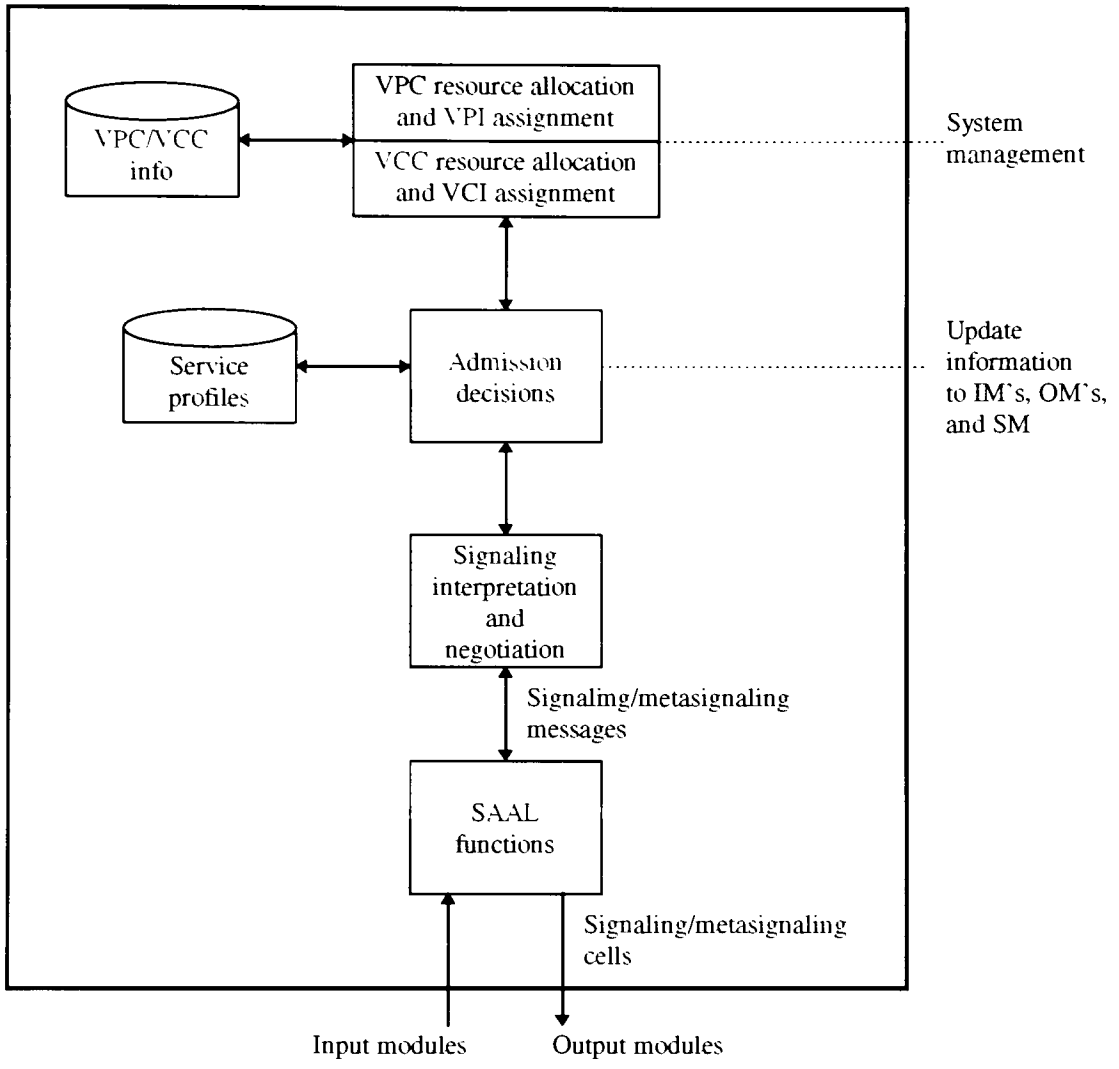


Figure 19 Functional block diagram of the CAC [1, p.176]

4.1.5 System Management

The system management portion of the ATM switch is important in that it helps to coordinate all of the other functional blocks within the switch, as well as monitoring and

adjusting all of the switch's operations. The amount of functionality given to the system management module is completely up to the designer of the switch. A switching system can operate correctly with only a limited amount of management functions. However, such a switch will not be as robust as one that provides the system administrator with the ability to closely monitor the switch's performance and configure the network in an optimal manner.

Although there is a wide variety of tasks that a system management module can be designed to perform, most of these will fit into a few general categories. For instance, the system management module must be able to supervise and coordinate all internal management activities, as well as communicating with users and system administrators. It is also necessary to collect and monitor certain management information. Finally, there are specific management responsibilities that must be carried out. A functional block diagram of a typical system management module is shown in Figure 20. The division of tasks shown here is only one possible design. A brief discussion of each piece of the system management follows.

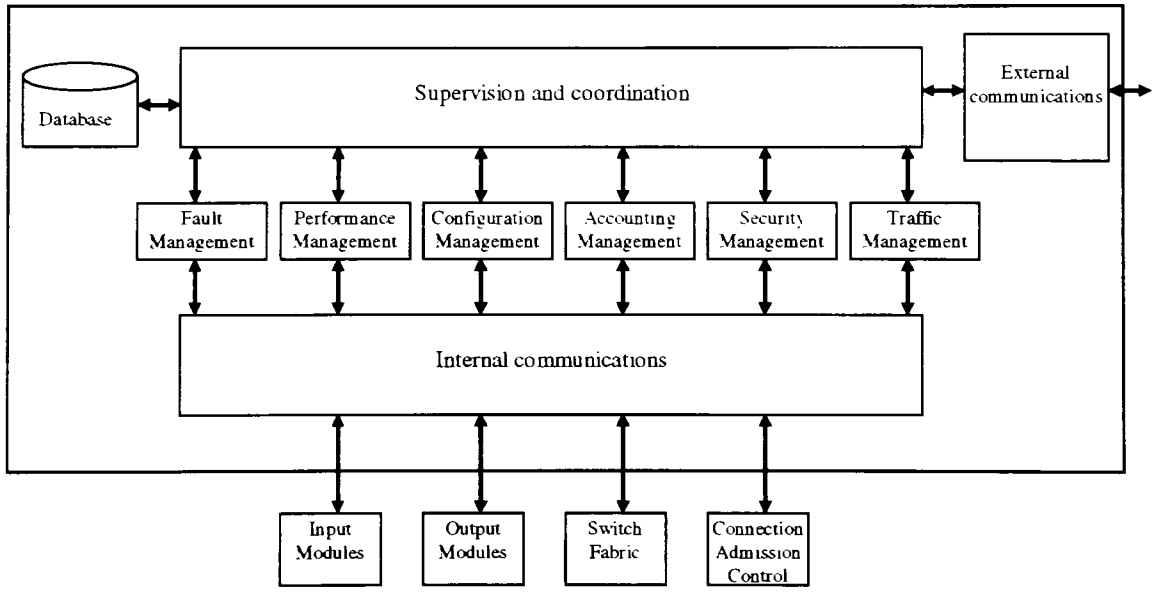


Figure 20 System management block diagram [1, p. 208]

The system management portion of and ATM switch must be able to recognize and process special operation and maintenance (OAM) cells. OAM cells can be used to serve many

purposes, including the detection and management of faults in both the physical and ATM layers. OAM cells can also be used to monitor performance. Parameters such as data transmission rates, number of cells lost, and number of cells containing errors can be documented by the system management module using OAM cells. System management functions are also required to properly configure different components of an ATM switching system.

Other important duties of the system management functional block include accounting management, which enables the switch to keep track of usage on the basis of individual connections. This information is useful if customers are to be billed based on the amount of network bandwidth they consume. It can also allow system administrators to better configure and plan a network.

If a network is to provide any security measures, they must be implemented by the system management module. Security functions are important for a variety of reasons. They are needed to prevent someone from assuming a false identity to gain access privileges. Security is also necessary to protect data from unauthorized modification or removal. Password protection, session control, redundancy of data, and encryption of data transmissions are several possible security features that can be designed into the system management of an ATM switch.

The system management portion of an ATM switch also has partial responsibility for managing traffic. It must regulate the flow of traffic into and out of the switch and attempt to prevent congestion. If congestion does occur, the system management must be able to react appropriately. The connection admission control (CAC) unit is in place mainly to prevent congestion. By allocating switch resources efficiently and ensuring that connections adhere to assigned quality of service parameters, the CAC can reduce the probability of congestion occurring.

However, if congestion is encountered within the switch, it is the system management's job to alleviate it. Possible areas of congestion must constantly be monitored. The switch fabric, cell buffers, input modules, and output modules are locations where congestion is

most likely to occur. If an unacceptable level of congestion is encountered, several possible actions can be taken. These include the selective discarding of cells, rerouting of cells, or notification of users and other network nodes to temporarily limit the flow of traffic.

4.2 Design of an ATM Switching System Architecture

Although each functional block of an ATM switch may have its own independent design, the architecture of the overall ATM switching system is dependant upon the placement and interconnection of each of these functional blocks. This section will examine several different options that need to be taken into consideration when designing a complete ATM switch. This includes a detailed discussion of how management and signaling functions are distributed and how data flows through the switch.

4.2.1 Flow of User Data

The most basic function of any ATM switch is to route cells from the input ports to the correct output ports. This process can be modeled using the simple block diagram shown in Figure 21.

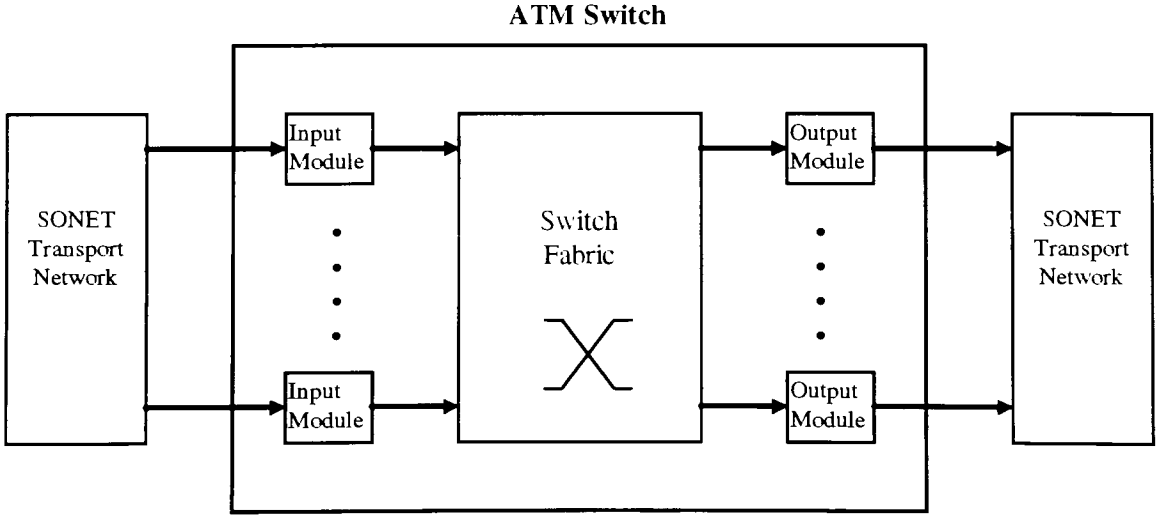


Figure 21 Flow of User Data Through an ATM Switch [1, p. 83]

The three main functional blocks of this model are the input modules, the switch fabric, and the output modules. A single input module is located at each of the input ports of the switch fabric. Each one is responsible for receiving frames from the SONET transport network and extracting the ATM cells from them. The input modules then prepare the cells and pass them to the switch fabric which routes them through the switch to the correct output port. At each output port, a single output module gathers cells and prepares them for transmission on the SONET network.

This model demonstrates the basic buffering and routing functions conducted by an ATM switch. Since the manner in which user data flows through an ATM switch is so straightforward, there exist practically no accepted alternative designs that will increase performance without drastically increasing complexity. However, there are many different internal implementations for the switch fabric, input modules, and output modules. Although designers are generally limited when planning the flow of user traffic through an ATM switch, there are several options that must be considered when designing data paths for the flow of control and management information.

4.2.2 Flow of control information

Within an ATM switch, control functions are necessary to establish and maintain all switched virtual circuits that pass through that switch. Necessary control information is transmitted through the switch using special control cells. The switch can distinguish these cells from other cells by the values stored in the VPI and VCI fields of the cell headers. When the switch encounters a normal user data cell, it routes the cell to the correct output port without examining the contents of the payload. However, after recognizing incoming cells containing control information, an ATM switch must be able to separate these cells from the stream of user traffic and process the information that they contain. If the switch needs to generate its own control information, it must produce the necessary control cells and mix them with the outgoing user data cells.

One possible model for handling the flow of control information through an ATM switch is shown in Figure 22.

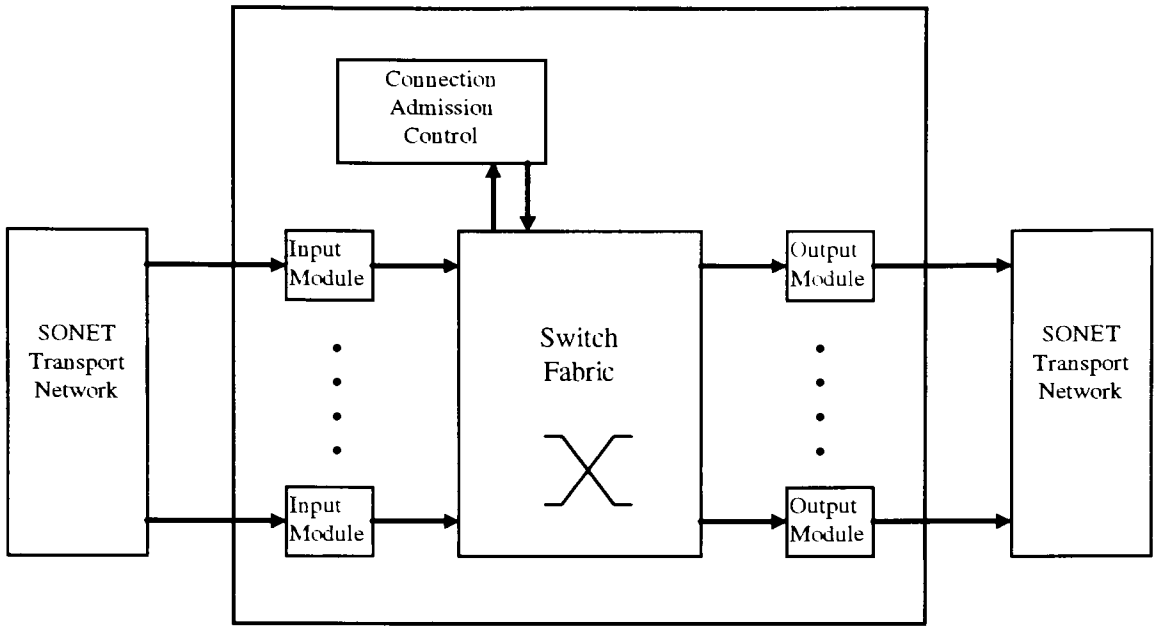


Figure 22 Flow of control information using switch fabric [1, p. 84]

This model contains all of the same functional blocks as the user data flow model, but it also adds the connection admission control (CAC) as an additional functional block. As discussed earlier, the CAC is responsible for negotiating new connection requests with users, deciding whether to admit or reject each connection, and allocating the appropriate network resources.

In the model shown in Figure 22, the CAC communicates only with the switch fabric. Therefore, it becomes the responsibility of the switch fabric to recognize control cells, separate them from user data cells, and route them to the CAC. All control cells generated by the CAC are passed to the switch fabric where they are then routed to the correct output ports.

However, if the alternate design shown in Figure 23 is used, it is not necessary for control cells to pass through the switch fabric at all.

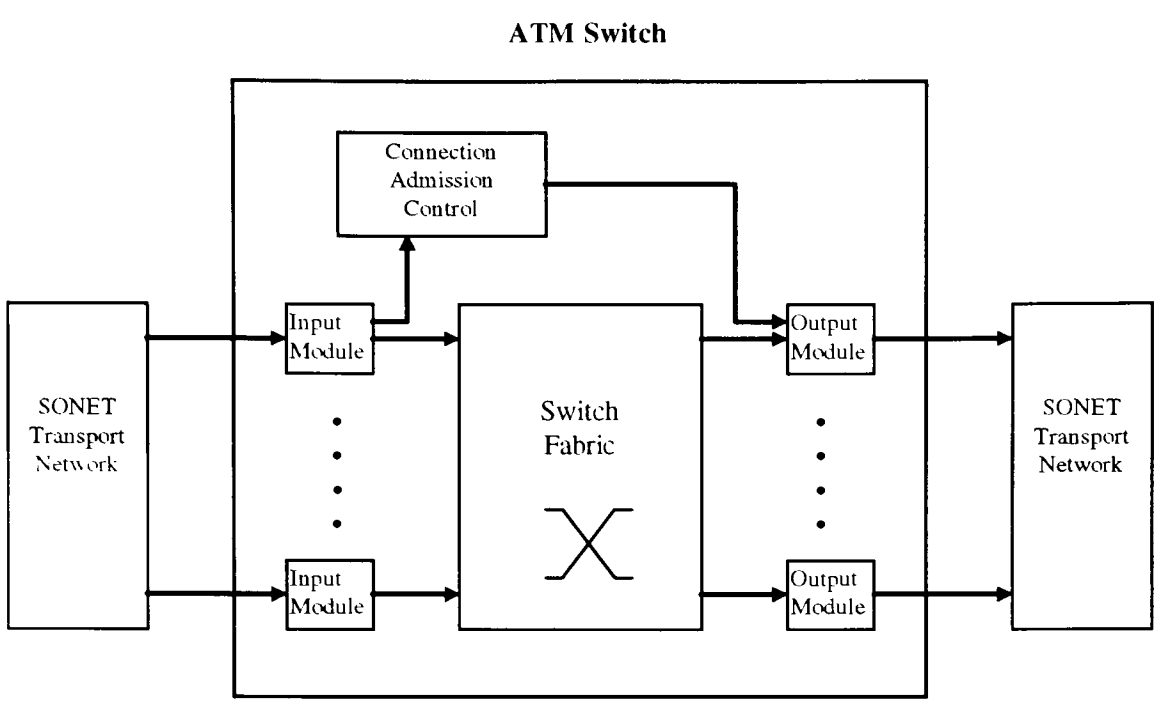


Figure 23 Flow of control information without using switch fabric [1, p. 84]

In this model, the CAC communicates only with the input and output modules. A direct connection is required between each of these modules and the CAC. Therefore, control cells bypass the switch fabric completely. In order for this to be accomplished, the task of detecting and routing control cells becomes the responsibility of the input module instead of the switch fabric. Furthermore, this design requires that the output modules be able to accept control cells from the CAC and mix them with outgoing user data cells.

There are both advantages and disadvantages to each of these models that must be considered before choosing an appropriate design for a particular switch. The main difference between these two approaches is the location of the extra complexity associated with identifying and routing control cells. If the model in Figure 22 is used, the processing overhead of scanning each cell's VPI and VCI fields in search of a control cell must be performed by the switch fabric. In the second model, this task is the responsibility of the input modules.

One main advantage of not using the switch fabric to handle control cells is the fact that the switch fabric is a critical component in the buffering and routing of user data traffic. Any extra cells that are handled by the switch fabric can degrade its performance. If control cells are handled by the switch fabric, they can occupy valuable buffer space. In addition, if control cells are passed to the CAC directly by the input modules, they can be processed and routed to the correct output ports in parallel as the switch fabric continues to deliver user data cells.

However, there are also disadvantages associated with having the input modules handle the control cells. The added complexity required to have the input modules recognize these control cells is greater than if the switch fabric were to perform the same task. This is because the switch fabric must already examine the VPI and VCI fields of each cell header in order to correctly route that cell. In addition, when the input modules communicate directly with the CAC, the output modules must also be designed to communicate with the CAC. This adds extra functionality to the output modules as well. This added complexity must be duplicated in each of the input and output modules.

Finally, it is advantageous to use the switch fabric to handle control cells since the switch fabric already contains the resources necessary to route these cells from any of the input ports and to each of the output ports. If the switch fabric is not used, then additional data paths must be added from each of the input modules to the CAC, and from the CAC to each of the output modules. Depending on the constraints of the switch design, these extra data paths may not be acceptable.

Alternatively, an ATM network can be designed to handle control information using an altogether separate common channel signaling system. In this scenario, the CAC needs to communicate directly with that signaling system, and will receive control information from there rather than from special control cells. However, since this issue is related to the design of an entire ATM network and not an individual switch, it will not be discussed further.

4.2.3 Flow of management information

The system management portion of an ATM switch is responsible for ensuring that the network is working correctly and that an acceptable level of performance and efficiency is being maintained. This is accomplished by monitoring and processing certain management information throughout the network. Special cells, referred to as operations and maintenance (OAM) cells, are used to carry this information from one node to the next. These management cells are handled by an ATM switch in a manner similar to control cells, however they must be processed by the system management functional block of each switch rather than the CAC. Management cells are also distinguished from user data cells and signaling cells by the values stored in their VPI and VCI fields. It is necessary for the switch to recognize these cells, separate them from all other types of cells, and route them to the system management module. After being processed, any outgoing management cells must be mixed in with the outgoing user data traffic.

When designing the flow of management information through an ATM switch, there are two general approaches. Similar to the way in which control information can be transmitted through a switch, one of these designs uses the switch fabric to handle the management cells, and one does not. Figure 24 shows a functional block diagram of the design that routes the management cells to the system management module using the switch fabric.

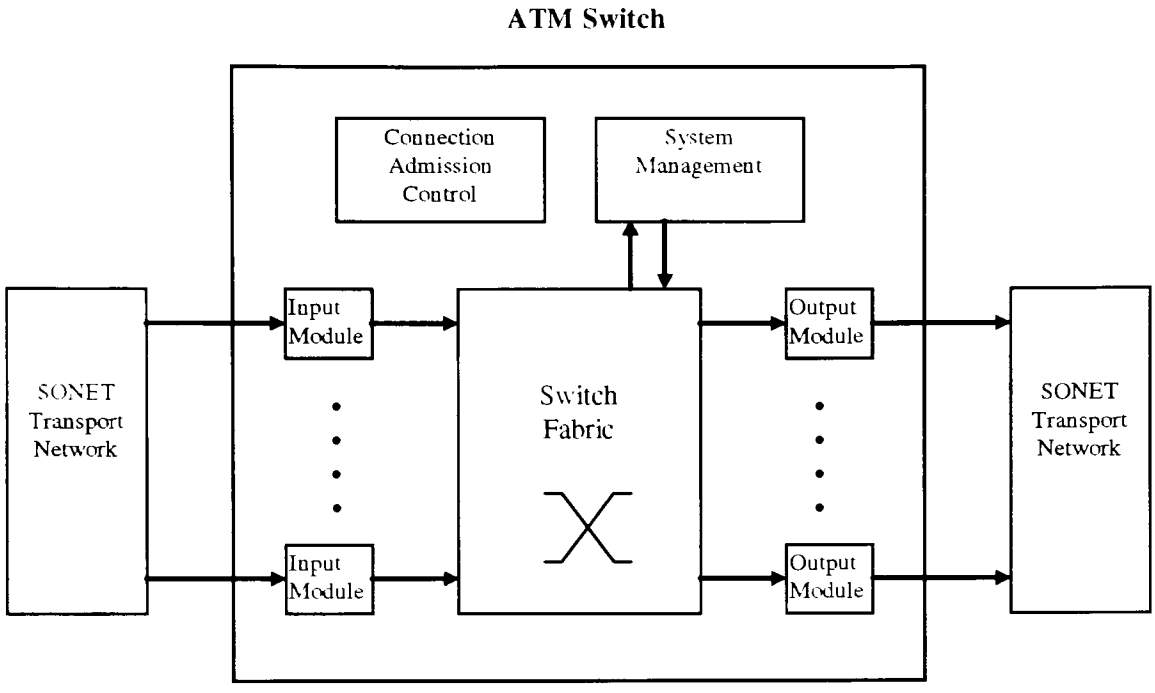


Figure 24 Flow of management information using the switch fabric [1, p. 86]

In this model, all OAM cells enter the switch fabric along with user data cells. The switch fabric must be able to identify these management cells and route them to the system management functional block where they are then processed. Any further OAM cells generated by the system management module are passed back to the switch fabric where they are then routed to the correct output ports in order to be transmitted over the SONET transport network along with user traffic. This model requires data paths that provide bidirectional communication between the switch fabric and the system management module.

The other option for the design of the flow of management information through an ATM switch is shown in Figure 25.

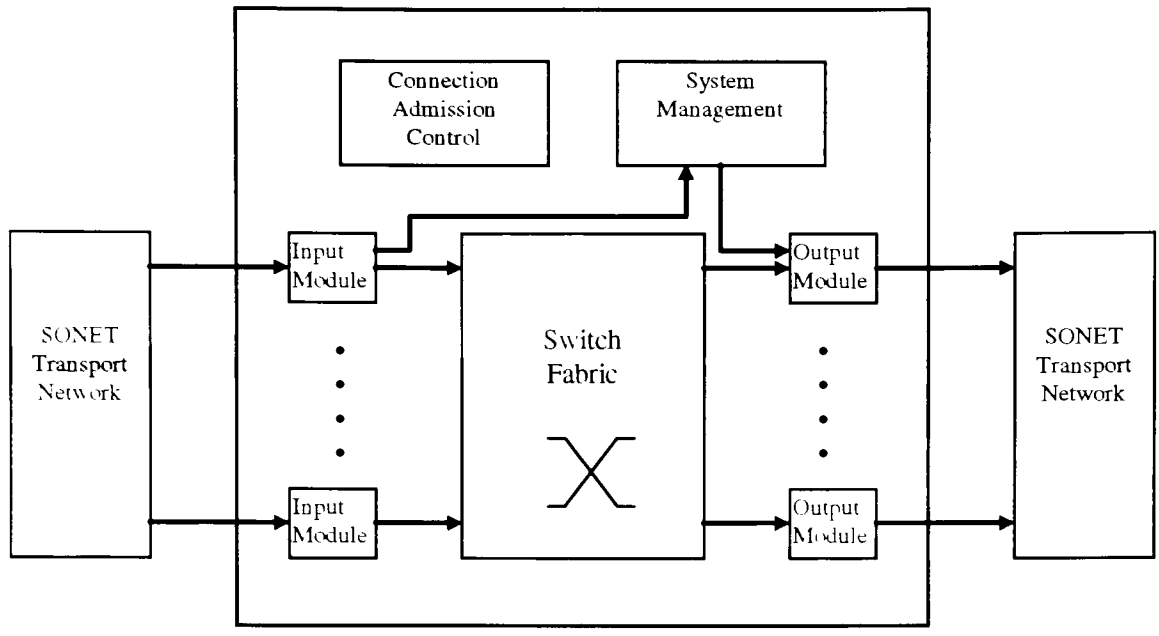


Figure 25 Flow of management information without using the switch fabric [1, p. 86]

In this design, OAM cells are recognized by the input modules and routed directly to the system management functional block. In order for this to be possible, a data path needs to exist between each input module and the system management unit. In addition, each output module must also have a direct path of communication with the system management module.

The pros and cons of each of these two designs are exactly the same as the two designs associated with the flow of control information through the CAC. When the OAM cells are handled by the input and output modules, it relieves the switch fabric of some possible congestion and allows it to process user data cells faster. However, routing the OAM cells without using the switch fabric requires the inclusion of extra data paths, and also increases the complexity of each of the input and output modules contained in the switch. These are just some of the tradeoffs that must be considered when designing the way in which management information flows through an ATM switch.

4.2.4 Distribution of connection admission control functions

In the switch architecture models examined so far, it has been assumed that all of the connection admission control functions are centralized into a single functional block. Every control information cell is passed to this CAC unit where all processing related to connection admission control for the entire switch is performed. When a centralized CAC is used, the other functional blocks of the switch are only responsible for ensuring that the CAC receives each cell that contains control information.

However, it is also possible to distribute these functions to different areas of the switch in order to effect the overall switch performance. One popular approach is to divide the input modules into several groups and have a separate CAC for each of these groups, as shown in Figure 26.

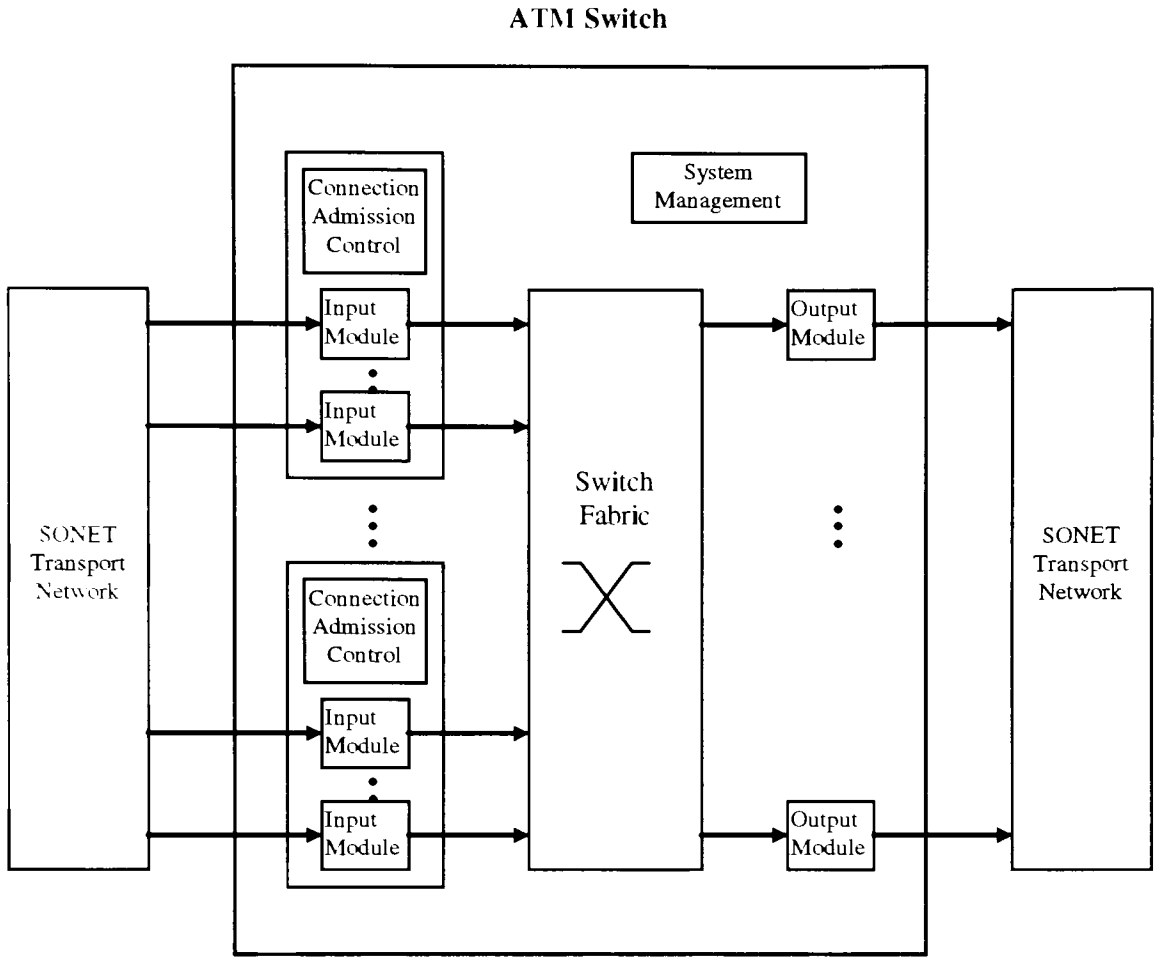


Figure 26 CAC functions distributed to blocks of input modules [1, p. 91]

Using this model, each input module is responsible for detecting cells containing control information and passing them to the associated CAC unit. Each CAC is identical to the others, and each performs the same operations as the centralized CAC discussed earlier. However, using this approach, each CAC is required to handle a smaller number of input ports. This is the key to obtaining performance improvements over switches using a centralized CAC. With the centralized approach, it is possible for the CAC in switches with a large number of input and output ports or heavy volumes of traffic to become overloaded. Distributing the CAC functions to blocks of input modules allows control processing to be performed in parallel. Since each CAC unit is required to communicate with a smaller number of input modules, the potential for a bottleneck is reduced.

The main drawback associated with distributing the CAC functions to blocks of input modules is the increased complexity that is added to the switch. In order to correctly and effectively perform control functions, each CAC must have access to information regarding the established connections and resource allocation of all other CAC's. This will require an extra degree of functionality for each CAC, and it will also require additional data paths between the CAC's allowing them to communicate with one another. Furthermore, distributed CAC requires several copies of the same functional block to be placed inside each switch, thus increasing the area of the chip on which it is fabricated.

It is possible to distribute the CAC functions throughout the switch even further as shown by the block diagram in Figure 27.

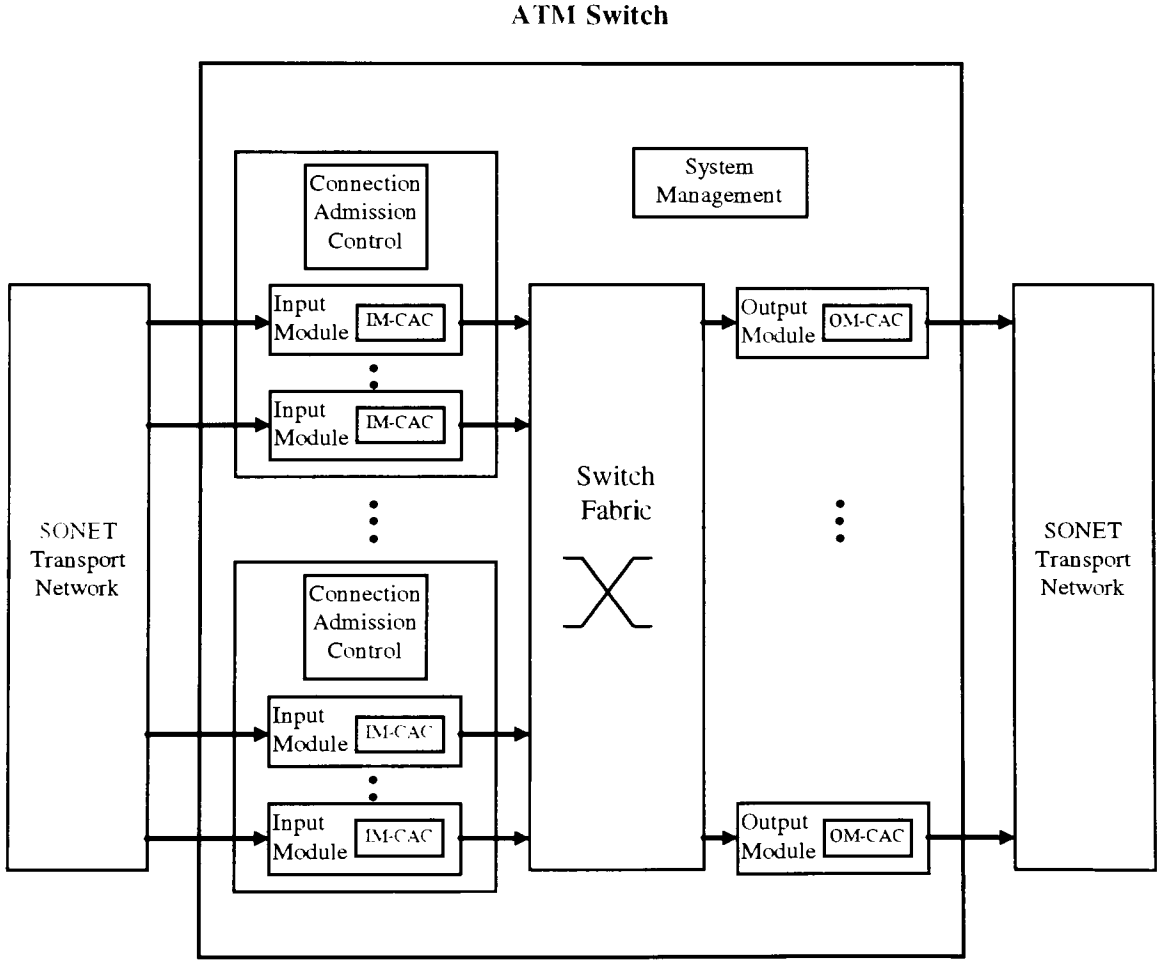


Figure 27 CAC functions distributed to input and output modules [1, p.93]

This implementation places some of the processing necessary for connection admission control tasks inside each of the input and output modules. The input modules are still organized into groups with a separate centralized CAC associated with each group. However, each input module also contains its own CAC functional block that can perform preliminary processing of control information before these cells are delivered to the appropriate CAC unit. These IM-CAC's are often responsible for extracting the signaling information from all control cells that it encounters. By processing and removing the overhead information attached to each control cell, the input modules reduce the amount of work that their CAC unit must perform. This allows control information to be processed faster and helps to avoid congestion.

Each output module also contains some internal CAC functionality. These internal CAC blocks are generally responsible for handling high-layer control information passed to them by each of the CAC's. The output modules must encapsulate this information into the format corresponding to a control cell so that it can be properly recognized by the next ATM switch to which it is transmitted. By allowing both the input modules and output modules to perform some of the CAC processing, the size of each CAC is reduced and control information is able to be interpreted much quicker. However, as the CAC functions are further distributed throughout the switch, the overall complexity of the design is increased.

4.2.5 Distribution of system management functions

System management functions within an ATM switch can be distributed using an approach similar to the distribution of connection admission control functions. A centralized system management unit, as discussed earlier, is relatively simple to implement. However, a single system management module is a potential bottleneck in a large switch that must process heavy volumes of traffic. A distributed system management approach, as shown in Figure 28, is one solution to this problem.

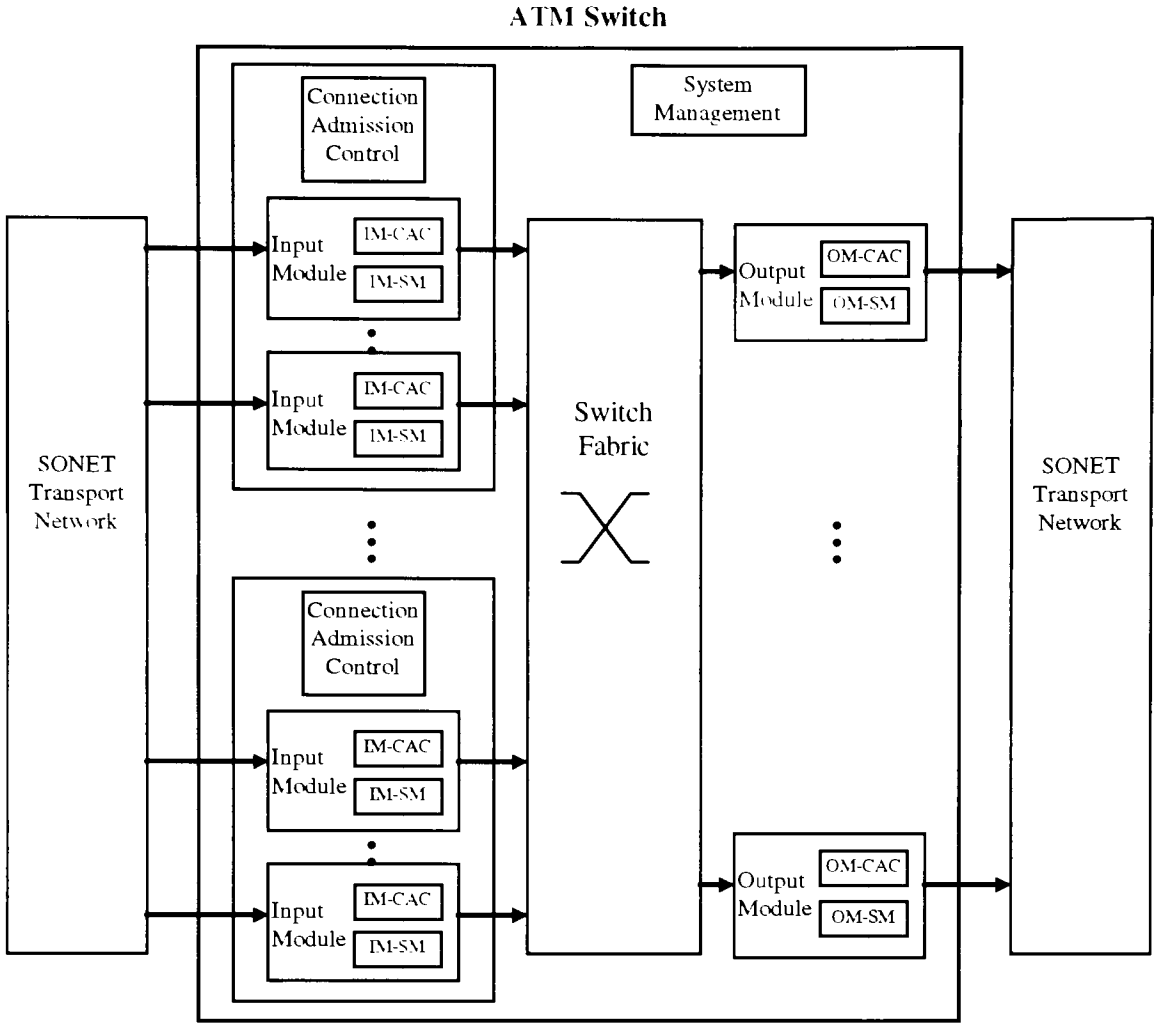


Figure 28 Distribution of system management functions to input and output modules [1, p. 94]

The model shown in this functional block diagram uses a single system management unit to perform the main processing of system management information. However, part of the system management functions are distributed to each of the input and output modules. Since many system management functions require that the frequency and quantity of incoming user data cells be monitored, it as more efficient to have this task handled by the input modules. The input modules can also extract the higher level system management information from all incoming OAM cells and pass this information directly to the main system management unit. Having the input modules preprocess each of the OAM cells relieves the main system management unit from having to perform this repetitive, time consuming task. The output modules are also responsible for monitoring the outgoing user data cells for system management purposes.

There are many other ways in which both the connection admission control and system management functions of an ATM switch can be distributed. The amount of distribution used in a design will have a definite effect on switch performance. A higher degree of distribution will reduce the probability of congestion that can be caused by a centralized processing unit. It will also enhance switch performance by allowing many functions to be performed in parallel. However, as the CAC and system management functions are further distributed throughout a switch, the complexity of the design also increases. Each of the distributed processing units must be able to communicate with the others in order to properly coordinate their actions.

There are endless possibilities to the manner in which the functionality of an ATM switch can be partitioned and distributed throughout the system. The designs studied here are just a few possibilities. Each one has its own advantages and disadvantages that must be carefully considered by the designer before a complete ATM switch architecture can be developed.

5 Input module

The input module is the functional block inside of an ATM switch that is responsible for providing an interface between each input port of the switch and the transport network that is used to carry the ATM cells from one switch to the next. This work will examine only input modules designed to interface with a SONET network. The input modules of a switch perform several important functions. They are responsible for terminating the physical layer signal and extracting the user data that is contained within each SONET frame. In addition, the boundaries between ATM cells in the incoming bit stream must be determined. Furthermore, each cell header must be processed and checked for errors before the user data can be routed through the switch fabric. Depending on the overall design of the switch, the input modules may also be responsible for recognizing and routing special signaling and management cells. It may also be necessary to have the input modules perform some of the system management and connection admission control functions.

5.1 Design of an input module

A functional block diagram of a generic input module is shown in Figure 29.

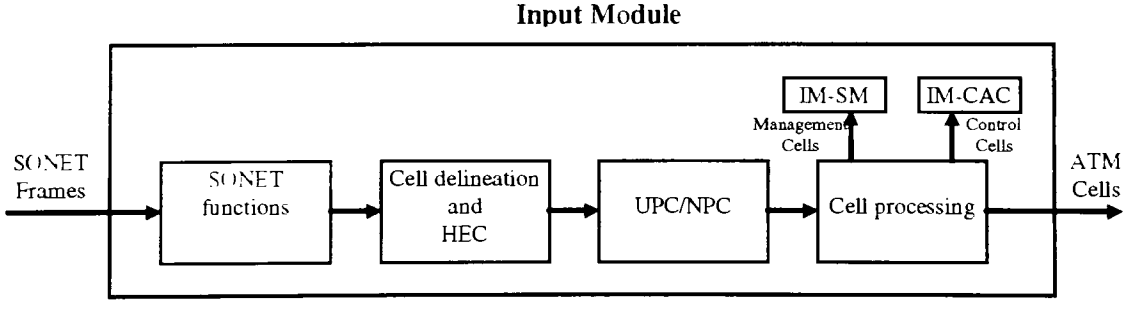


Figure 29 Functional block diagram of an input module [1, p. 98]

As illustrated in the diagram, the functions of an input module can be divided into six separate units. The necessary physical layer functions are handled by the SONET and cell delineation blocks. The UPC/NPC and cell processing blocks are responsible for providing

the ATM layer functionality. The IM-SM and IM-CAC units are the portions of the system management and connection admission control that may reside within each of the input modules, but they are not absolutely necessary.

5.1.1 SONET functions

The first job of the input module is to receive SONET frames on each incoming link and extract the ATM cells from them. In order for this to occur correctly, the overhead associated with each SONET frame must be processed in accordance with the SONET protocol. Some of the main SONET related functions performed by each input module include the recovery of each incoming frame, payload mapping, demultiplexing, and frequency justification/pointer processing. The main purpose of the SONET functional block in an input module is to correctly extract the payload from each frame and pass it to the cell delineation and header error control block where it will be processed further. The exact manner in which ATM cells are mapped into SONET frames is covered in detail by Chapter 3.

5.1.2 Cell delineation and header error control

The second functional block within each input module can be referred to as the cell delineation unit. This block is responsible for three important physical layer tasks. These include header error control, cell delineation, and cell rate decoupling. The header error control process is necessary to ensure the integrity of each cell's header information. Cell delineation is the process by which cell boundaries within the incoming data stream are determined. Finally, cell rate decoupling is necessary to remove all idle cells from the incoming bit stream.

The cell delineation and header error control functional block must also continually monitor certain information that is needed by the system management portion of the ATM switch. This information can include important statistics such as the number of incoming cells that contained errors, the number of these errors that were corrected, and the number of cells

that were discarded. This information must be relayed to the system management module where it will be processed. Each of the functions associated with the cell delineation and header error control block will now be examined in further detail.

5.1.2.1 Header error control

Each ATM cell includes a header error control (HEC) byte located in the fifth byte of the header. This HEC byte is used as a CRC code to ensure that the other four bytes of the cell header have been received correctly. The information contained within the header is vital to the correct delivery of each ATM cell. Single bit errors can be corrected by the HEC mechanism, but cells found with multiple bit errors are dropped by the switch. Only the header of each ATM cell is checked for correctness. A higher level protocol is responsible for checking the validity of the user data.

Each time an ATM cell is ready to be transmitted, the HEC byte is generated after the other four bytes of the header have been established. In order to generate this value, the HEC field is first initialized to all zero. Each of the 40 bits of the entire five-byte header is now used to represent a coefficient of a 39-degree polynomial. This polynomial is then divided (modulo 2) by the generator polynomial $x^8 + x^2 + x + 1$. This generator polynomial has been established by ATM standards to produce the 8-bit CRC code. This division of the two polynomials will result in an 8-bit remainder. Next, an 8-bit coset pattern of 01010101 is added to this remainder. This pattern of alternating zeros and ones is used to improve the performance of the cell delineation process when a bit-slip occurs. Finally, the resulting byte replaces the all zero HEC field in the cell's header.

In order to validate the correctness of a cell's header, an input module must first subtract the coset 01010101 from the HEC field. The complete five byte header is again converted into a 39-degree polynomial which is then divided by the same generator polynomial that was previously used to calculate the HEC value. The resulting 8-bit remainder is referred to as the syndrome. If a syndrome of zero is obtained, it is an indication that there are no detectable errors within the cell's header. However, a nonzero value signifies that the cell header contains errors or that the cell boundaries are not correctly established. [1, p. 125]

5.1.2.2 Cell delineation

The calculation of syndromes is also used to determine the boundaries between newly arriving cells. Figure 30 provides a state diagram for the cell delineation process.

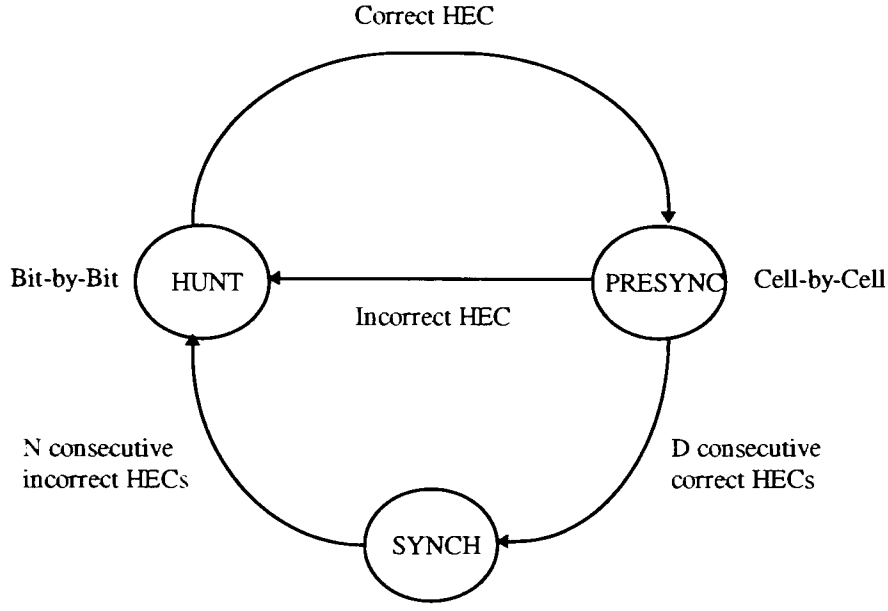


Figure 30 Cell delineation state diagram [1, p. 126]

The cell delineation process begins in the HUNT state. The incoming bit stream is examined using a 5-byte sliding window. At each position, the window is assumed to contain a cell header. Therefore, the fifth byte located within the window represents a HEC value. The syndrome for that header is calculated as discussed earlier. A nonzero syndrome indicates that a valid header was not yet located. Therefore, the window is shifted by one bit and the syndrome is again calculated. This process repeats until a syndrome of zero is discovered. At that point, it is assumed that a valid header has been located and the PRESYNC state is entered.

In the PRESYNC stage, the incoming byte stream is scanned on a cell-by-cell basis. The header of each cell is checked to see if a syndrome of zero is calculated. If D consecutive

cells have a zero syndrome, then the input module concludes that the cell boundaries have been correctly located. The cell delineation process then enters the SYNC state. However, if an invalid header is encountered during the PRESYNC state, the process returns to the HUNT stage.

Once the SYNC state has been entered, the cell delineation functional block has three primary tasks. The first of these is to locate any cells with uncorrectable header errors and discard them. In addition, cell boundaries must be constantly checked to ensure that they are properly maintained. If N consecutive cells with nonzero syndromes are encountered, the cell delineation process returns to the HUNT state in an attempt to reestablish synchronization. Finally, the SYNC state is also responsible for performing cell rate decoupling and passing all nonempty cells with syndromes of zero to cell descramblers.

5.1.2.3 Cell rate decoupling

Since SONET is a synchronous protocol, it must transmit data during a series of periodic intervals.. Due to the fact that ATM is an asynchronous technology, it may not have information to send during each SONET time slot. Therefore, these unused time slots must be filled with idle cells in order to satisfy SONET protocol requirements. When a SONET frame is received by the input module, each of these unassigned cells must be identified and removed from the incoming bit stream. This cell rate decoupling is performed during the SYNC state of the cell delineation process.

5.1.3 UPC/NPC

The usage parameter control (UPC) portion of an input module is responsible for monitoring and enforcing traffic rates across a user network interface (UNI). Similarly, the network parameter control (NPC) portion performs the same functions across a network to network interface (NNI). The UPC/NPC functional block must be designed to enforce a particular traffic policing algorithm. A functional block diagram of the UPC/NPC module

is shown in Figure 31. Each component of the UPC/NPC module will be discussed further.

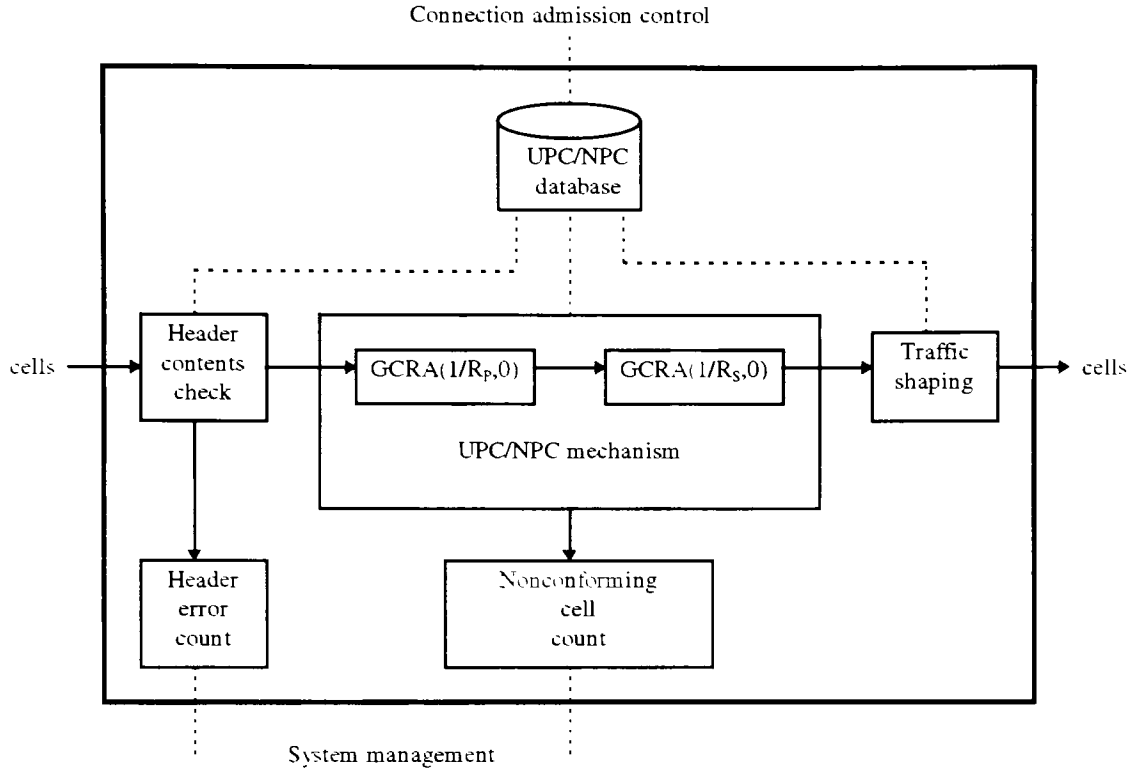


Figure 31 Functional block diagram of UPC/NPC [1, p. 127]

5.1.3.1 UPC/NPC database

The UPC/NPC database is in direct communication with the connection admission control unit, whether it resides within the input module or elsewhere within the switch. The database receives and stores information pertaining to active virtual connections. Traffic parameters for each VPC/VCC pair that needs to be monitored is gathered. This information includes such parameters as peak cell rate, sustainable cell rate, and burst tolerance. This data is necessary in order to verify that each of these virtual connections is complying with previously established restrictions on traffic flow.

5.1.3.2 Header contents check

Immediately upon entering the UPC/NPC unit, a cell's header is checked to ensure that each of its fields contains valid data. Since a header error check has already been performed on each cell, it is now necessary to verify that the information that is contained in the header makes sense. This portion of the UPC/NPC typically scans a header for invalid VPI/VCI values or undefined values in other header fields. A simple check is performed, and all cells that do not pass are removed from the stream of valid cells that are allowed to pass through the input module.

5.1.3.3 Header error count

Only cells that are rejected by the header contents check are sent to the header error count functional block. Such cells are generally counted, and the contents of their headers are stored. If the count of invalid cells exceeds a predetermined amount, the system management unit may request the stored header values from the header error count module. This information can then be analyzed in an attempt to find and correct any potential performance problems.

5.1.3.4 UPC/NPC mechanism

The UPC/NPC mechanism is the core of the UPC/NPC functional block. This is where the actual monitoring and enforcing of traffic rates occurs. A count of the number of cells carried over each virtual connection is also maintained by the UPC/NPC mechanism. This is necessary information if a switch is to support account management features. The traffic management algorithm employed by the UPC/NPC can be different for each virtual connection. Cells that do not conform to the algorithm in use are counted and discarded.

ATM standards specify the generic cell rate algorithm (GCRA) for use in the UPC/NPC. The GCRA defines the permitted cell flow on a connection and ensures that each connection conforms to assigned bandwidth constraints. This algorithm is similar to a dual

leaky bucket to monitor both the peak cell rate (R_P) and sustainable cell rate (R_S) of a virtual connection.

5.1.3.5 Traffic shaping

The traffic shaping functional block allows an input module to modify the stream of cells that is being sent to the switch fabric. Traffic shaping is a process that eliminates bursty traffic by smoothing out the incoming cell stream. This is accomplished by temporarily storing cells in internal buffers and then spacing them more evenly on the outgoing data stream. Of course, the effectiveness of traffic shaping is limited by the size of the internal buffers being used. However, this optional technique can aid in preventing congestion from occurring inside the switch.

5.1.4 Cell processing

The cell processing functional block of the input module is necessary to identify and separate signaling and management cells from the stream of user data cells. The cell processing module examines the information contained in each cell's header. If necessary, signaling cells are routed to the connection admission control (CAC) unit, and management cells are sent to the system management (SM) unit. Translation of the VPI and VCI fields of cells containing user data is performed so that they can be correctly routed through the switch fabric. The cell processing block can also attach an internal tag to each cell before it is passed to the switch fabric. A switch designer can use an internal tag to keep track of information that may be useful elsewhere in the switch. Such a tag is later removed from the cell before it leaves the switch. A functional block diagram of an input module's cell processing unit is shown in Figure 32.

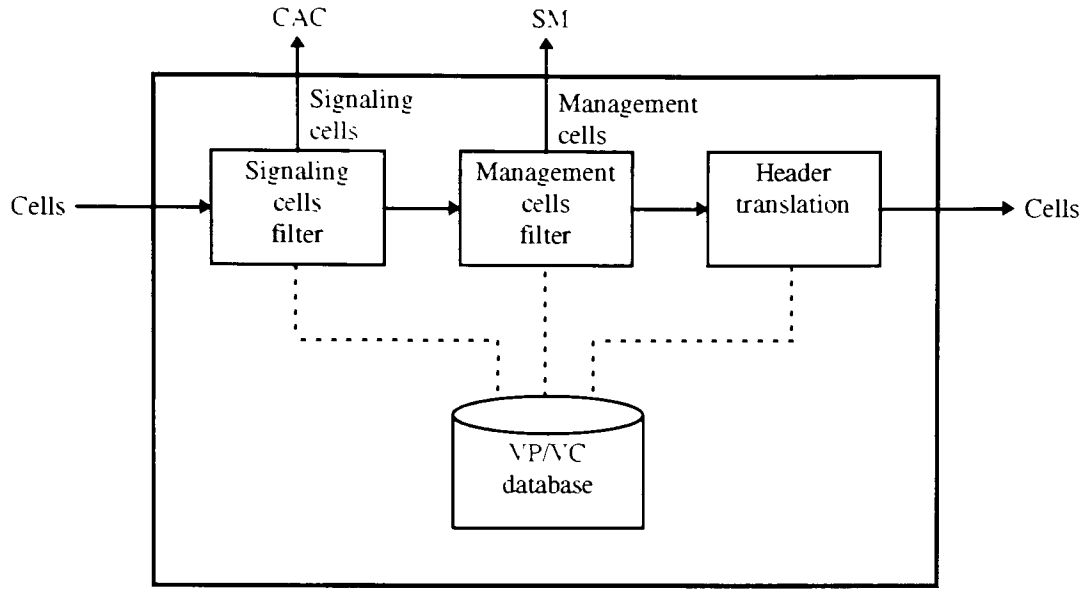


Figure 32 Cell processing functional block diagram [1. p.129]

The VP, VC database contains a lookup table that is necessary to perform header translation. For each active virtual connection, the database contains the corresponding VPI and VCI values associated with the outgoing link. The database can also be used to determine which VPI VCI pairs are already in use, which are reserved for signaling and management functions, and which belong to a broadcast or multicast connection. Furthermore, additional information about each virtual connection may also be stored in the VP/VC database. This can include the identities of both the source and destination users, the output port, cell delay and cell loss tolerances for the connection, and any delay priorities.

Cells that enter the processing stage of an input module will first encounter a filter to detect signaling cells. These cells can be identified by comparing their VPI and VCI values with those stored in the VP/VC database. When a signaling cell is encountered, the filter must determine whether or not that cell needs to be processed by the CAC. If processing is required, the signaling cell is removed from the data stream and routed to the CAC along the appropriate path. All other cells are allowed to pass through the filter unaltered.

The management cell filter performs a similar function on incoming management cells. The filter recognizes all management cells by their VPI and VCI fields. If a management cell requires special processing, it is routed to the system management unit of the switch. However, the management cell filter differs from the signaling cell filter in that it may also be required to examine user data cells. When requested by the system management unit, the management cell filter needs to be able to monitor the performance of cells containing user data and report the results back to the SM. This task can consist of counting and performing error checks on block of user data cells.

The final task involved in processing cells that pass through an input module is that of header translation. Header translation is necessary because a single virtual connection may have different VPI and VCI values on different physical links. This task is performed by the header translation block by simply looking up the new VPI/VCI values in the VP/VC database. The previous VPI and VCI values in the cell header are then replaced with the new values before the cell leaves the input module.

5.1.4.1 IM-CAC and IM-SM

The presence and responsibility of any connection admission control or system management functions within an input module is dependent on the degree to which these functions are distributed through the entire switch. A switch utilizing a completely centralized approach to both CAC and SM function will not require the presence of either an IM-CAC or a IM-SM. In this case, the input module must route the signaling and management cells directly to the appropriate locations.

5.2 C++ model

One intent of this thesis is to create a functional model of an input module using the VHDL design language. In order to ensure the proper functionality of this model in simulation, it was necessary to also develop a model of an input module using an object oriented language such as C++. A C++ model allows the behavior and proper functionality

of an input module to be duplicated without requiring details of the underlying hardware implementation. Identical sets of test cases can be passed to both the C++ and VHDL models of the input module. If any discrepancies between the results are encountered, it is an indication that the model is not correct. The test results can also be used to track down and correct the problem.

Using an object oriented approach to the software model of an input module is advantageous in that it allows much of the code to be reused when modeling an output module. The entire C++ model of an input module consists of the following files: "ATMCell.cc", "ATMCell.h", "SONETFrame.cc", "SONETFrame.h", and "input.cc". A "makefile" is also provided to aid in the compilation of the model. All of the C++ code used to create this model is included in Appendix E. The purpose of each of the files will now be discussed in detail.

5.2.1 ATMCell.cc and ATMCell.h

The "ATMCell.cc" file contains the class declarations and member functions necessary to establish a C++ class that emulates the behavior of an ATM cell as it travels through a switch. This class, named "ATMCell", contains private data types that are used to separately store the header and payload information contained by the cell. There is also a private variable that is used to indicate whether a cell contains valid, assigned data or if it is just being used as an idle cell to occupy an unused time slot within a SONET frame. Member functions are included to alter the value of this variable, and to examine its contents.

In addition to being able to initialize and print out the contents of a cell, the "ATMCell" class has many other important functions that it can perform. These include the ability to read in the contents of a single cell from a file, or to write the contents of a cell out to a file. The "ATMCell" class also provides a function that allows the user to input the desired number of ATM cells to be generated. The percentage of these cells that are to be valid and not idle can also be specified. The contents of each cell will then be randomly generated and saved to a file.

These functions are necessary to automate the testing procedure as much as possible. By providing the ability to generate a specific number of cells, each with a certain probability of being assigned, the "ATMCell" class allows different traffic patterns to be simulated. By saving both the input and output streams to separate files, the exact same test cases can be applied to the VHDL model. This provides a simple method of verifying the functionality of both models.

Another important feature of the "ATMCell" class is its ability to perform header error checking on an ATM cell. The C++ model accomplishes this through means of a table lookup algorithm. Upon creation of a variable of type "ATMCell", a table containing all possible 8-bit error patterns is calculated. Whenever a syndrome value needs to be generated, it can simply be retrieved from the lookup table. This is necessary whenever an incoming cell needs to have its header checked for errors, or whenever an outgoing cell needs to have a new HEC value calculated.

5.2.2 SONETFrame.cc and SONETFrame.h

The file "SONETFrame.cc" contains the necessary variable and member function declarations to establish a C++ class that simulates the structure and behavior of a SONET frame. This class contains a single private variable that is used to store information according to the SONET STS-3c frame format.

The member functions included in the "SONETFrame" class provide such basic functions as clearing the contents of the entire frame, initializing the contents of the frame's payload with specified values, and printing the contents of the frame. It is also possible to store an entire frame in a file, or read in the contents of a frame from a file.

5.2.3 input.cc

The "input.cc" file contains the actual C++ source code used to model the functionality of an input module for an ATM switch. An executable version of this code can be created using the accompanying "makefile." The input module receives a stream of SONET frames as input. These frames must be stored in a file named "inmodframes.in" prior to execution of the model. The input module reads from this file, extracts the assigned ATM cells from the payload of each frame, performs the necessary header error checking, and outputs a stream of individual ATM cells. These cells are stored in a file named "inmodcells.out". All of these functions will be performed automatically by the input module. The user only needs to create the "inmodframes.in" file containing the desired input frames and run the executable version of the input module model. Each value in the files "inmodframes.in" and "inmodcells.out" are decimal numbers between 0 and 255 that are used to represent the value of a single byte. Examples of both the "inmodframes.in" and "inmodcells.out" files are provided in Appendix B.

5.3 VHDL model

The VHDL portion of this thesis is used to provide a functional model of an input module for simulation purposes. The correct operation of this model has been verified using the C++ model discussed in the previous section. The intent of this work is to provide a working model of one of the functional blocks of a complete ATM switch. This VHDL model of an output model can then be used as a tool in future research to further explore the theory associated with ATM switch performance. By integrating the output module with the other necessary pieces of a switch, it is possible to study the effects of different switch architectures on network performance.

All of the VHDL code used to model both the input and output modules is included in Appendix F. Throughout the development of this model, certain coding standards were adhered to. A suffix was attached to the name of each object in order to indicate the type of that object. The suffix "_s" indicates that the object being referred to is a VHDL signal. Signals that are also component ports are represented using the suffix "_p". An entity is

indicated using the "_e" extension, and an architecture name ends with a "_a". Packages and types are also differentiated using the "_Pkg" and "_Typ" suffixes respectively. Finally, all constants are denoted using "_c". Variables within a process do not have any extensions attached.

The VHDL model of an input module is contained within six separate VHDL files: "inoutmod_pb.vhd", "xor_ea.vhd", "shift_ea.vhd", "crc_check_ea.vhd", "input_module_ea.vhd", and "input_module_tb.vhd". All file names with a suffix of "_pb.vhd" indicate the file contains a package declaration and the associated body definition. Files with the suffix of "_ea.vhd" contain the entity and architecture declarations for a particular component. Finally, all files ending with "_tb.vhd" contain a testbench for a complete design. The contents and purpose of each of the VHDL files will now be examined in detail.

5.3.1 inoutmod_pb.vhd

This file contains a package declaration and body defining all of the types, functions, and constants that are used to create a model for both the input and output modules. Subtypes used to hold both an ATM cell and a SONET frame are declared. In addition, functions are provided to convert between characters and bytes as well as integers and bitvectors.

5.3.2 xor_ea.vhd

This file contains the entity and architecture definitions for a two input exclusive-or gate. An external view of the XOR gate is shown in Figure 33.

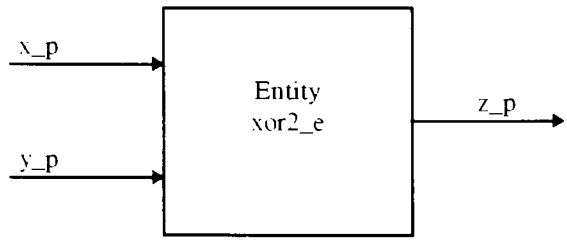


Figure 33 Two input XOR gate symbol

This entity has a behavioral architecture that performs the exclusive-or function on the input bits and outputs the resulting bit. This gate is used in the construction of a hardware entity used to check the validity of the header error control byte attached to each of the incoming ATM cells.

5.3.3 shift_ea.vhd

This file contains the entity and architecture pair required to model a one-bit shift register. An external view of this entity is shown in Figure 34.

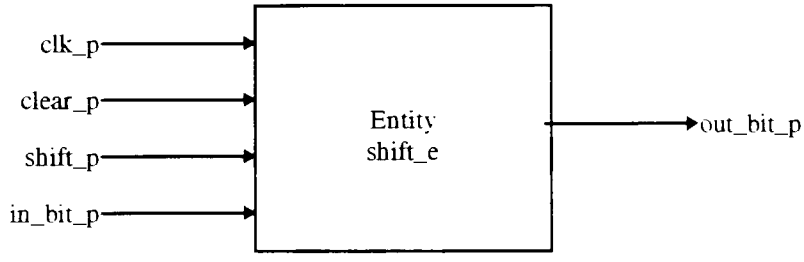


Figure 34 One-bit shift register symbol

The architecture associated with the "shift_e" entity is a behavioral one. All of the inputs to the shift register are active high. The "clear_p" signal is asynchronous and will store a '0' bit in the register. Whenever the "shift_p" signal is active during the rising edge of the "clk_p" signal, the input bit ("in_bit_p") will be shifted into the register. The current content of the register is output through the "out_bit_p" signal. Along with the exclusive-or gate, the one-bit shift register is used to build a structural model of the CRC checking hardware used by the input module.

5.3.4 crc_check_ea.vhd

The component instantiations and port maps necessary to create a structural model of the CRC checking hardware are located in the "crc_check_ea.vhd" file. An external view of this entity showing all of its ports is given in Figure 35.

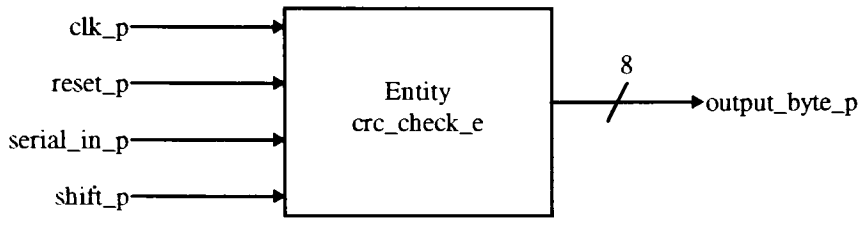


Figure 35 Cyclic redundancy checking hardware symbol

For each cell that it receives, the input module must verify that the header information has been transmitted correctly. It does this by executing the CRC algorithm and ensuring that the value stored in the fifth byte of the header, the header error control byte, corresponds to the first four header bytes. Prior to transmission of a cell, the output module generates the HEC byte by first multiplying the 32 bits of header information by 2^8. This serves the purpose of appending eight zero bits to the four original header bytes. A complete five byte cell header with the HEC byte zeroed out has now been obtained. Each bit of this 40-bit string represents a coefficient in a 39-degree polynomial. In order to calculate the HEC byte the output module must divide this 39-degree polynomial by the generating polynomial of $X^8 + X^2 + X + 1$ as defined by the ATM standards. This polynomial corresponds to the 9-bit pattern of 100000111. When the division is performed, an 8-bit remainder is generated. This value is the header control byte corresponding to the first four header bytes. The HEC byte is inserted into the fifth byte of the header before it is transmitted by the output module.

Upon receipt of a new cell, the input module divides the entire 40-bit header by the generating polynomial ($X^8 + X^2 + X + 1$). If the HEC value correctly matches the first four bytes of the header, the division will result in a remainder of zero. However, a non-zero

remainder indicates that the HEC value is incorrect and an error has occurred in the transmission of the cell's header.

The above algorithm for performing the necessary CRC checking within the input module has been implemented in the VHDL model using a dividing circuit consisting of three exclusive-or gates and eight 1-bit shift registers. A block diagram illustrating the internal operation of the "crc_check_e" entity is given in Figure 36.

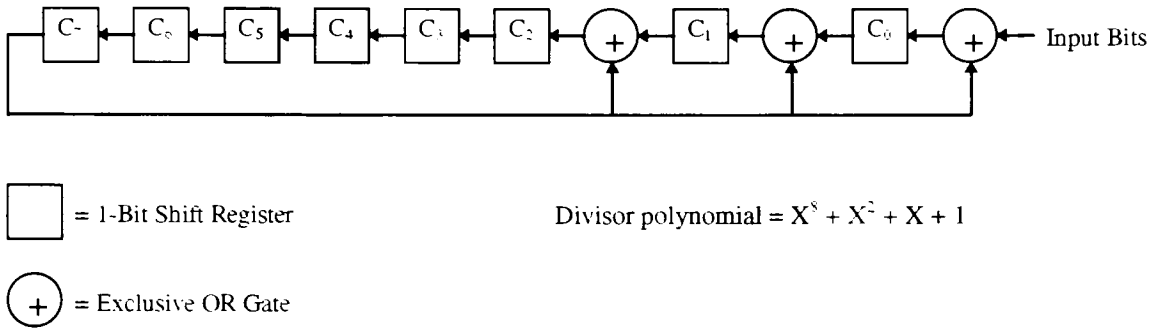


Figure 36 Block diagram of CRC circuit

The division is performed by first initializing each of the eight 1-bit shift registers to zero. This is accomplished by holding the "reset_p" signal high. As the input module receives a cell header, it shifts each bit into the "crc_check_e" entity using the "serial_in_p" and "shift_p" ports. If a complete five byte header has been shifted in and the "output_byte_p" port indicates a remainder of zero, the input module assumes that the header contains no errors. A step by step example of this process is provided in Appendix A.

5.3.5 input_module_ea.vhd

The file "input_module_ea.vhd" contains the top level behavioral model of an input module that is used to interface an ATM switch with a SONET transport network. An external view of the entity "input_module_e" that is defined in this file is shown in Figure 37.

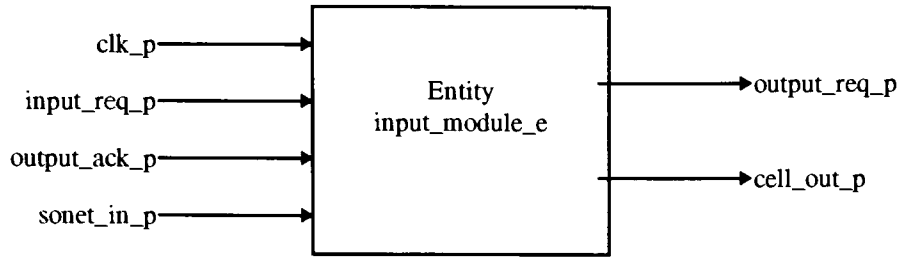


Figure 37 Input module symbol

A SONET transport network will transmit a complete frame every 125 μsec. Since each STS-3c frame contains 2430 · 8 = 19,440 bits, an input module will receive one bit every 6.43 nsec. Therefore, the "clk_p" signal must have a frequency of 155.52 MHz. During each clock cycle that the "input_req_p" line is asserted, an incoming bit is received through the port named "sonet_in_p". As bits are received, the input module shifts them into the internal "crc_check_e" entity according to the cell delineation algorithm. Once the ATM cell boundaries have been established within the incoming bitstream, these cells are extracted and placed on the "cell_out_p" port. Each time a valid cell is placed on this port, the "output_req_p" line is pulsed high. The cell will remain on the output port until the input module receives an acknowledgement on the "output_ack_p" input port that the cell has been successfully read.

5.3.6 input_module_tb.vhd

In order to test the functionality of the input module, a test bench has been created in the file "input_module_tb.vhd". This file instantiates a single input module component and generates a clock signal with a period of 6.43 nsec to drive it. The input module is tested through the use of two concurrent processes. The first process provides the input module with incoming SONET frames by reading them in from an ASCII file named "inmodframes.in". The exact contents of this file are transmitted to the input module one bit at a time. The second process contained within the testbench is responsible for reading ATM cells from the output side of the input module and storing them in an ASCII file named "inmodcells.out".

5.4 Testing procedure

In order to test the input module, the desired input data consisting of valid SONET frames must first be placed into the file "inmodframes.in". The testbench entity named "input_module_tb" along with the associated architecture "testbench_a" must then be loaded into the Mentor Graphics QuickVHDL simulator. The simulation must then be run for an amount of time great enough to process the entire input file. The output file "inmodcells.out" can then be examined to determine if the ATM cells were correctly extracted from the incoming SONET frames. The ASCII input and output files used by the VHDL model follow a format identical to those used by the C++ model. This allows the same input data to be supplied to both models. The output files can then be easily compared to locate any differences. Input files containing SONET frames were generated using the output produced by the C++ model of the output module. Appendix B contains a complete example of an actual test case that was run on the input module.

Numerous test cases were run on both the C++ and the VHDL model of the input module. In each case, the results of both models were compared to ensure that they produced the same results. Simple test cases to test the correct operation of the input module were created first. These test cases ensured that ATM cells were correctly extracted from the incoming SONET frames, particularly when one cell spanned the boundary between two frames. Simple tests were also performed to ensure that the cell delineation and header error control algorithms functioned correctly before larger tests were conducted.

Test cases were formulated to simulate several different traffic patterns that an input module might be encountered. In order to test heavy traffic, 100,000 consecutive valid cells were transmitted to the input module. Although this test case took several hours to complete, the results produced by both the C++ and VHDL models were identical. Other tests were also run in which idle cells were introduced. The percentage of these cells that were valid was also varied for different cases.

6 Output module

Once cells have been correctly routed through an ATM switch, they must be prepared for transmission over the next physical link. This job is the responsibility of the switch's output modules. A single output module is connected to each output port of the switch fabric. An output module is similar to in input module in that it performs many of the same functions in reverse. Once the ATM cells have been properly prepared for transmission, the output module must map them into SONET frames.

6.1 Design of an output module

The design of an output module is simpler than that of an input module because it has less tasks to perform. A possible functional block diagram for an output module is shown in Figure 38. Each portion of the output module will be discussed in further detail.

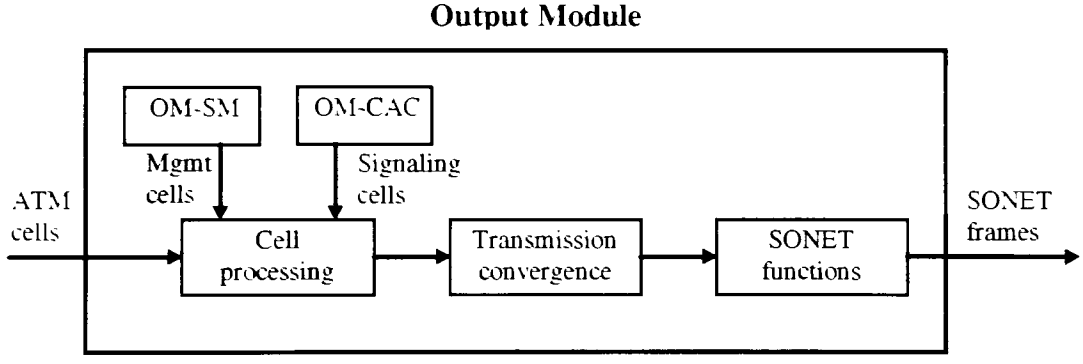


Figure 38 Function block diagram of an output module [1, p. 134]

6.1.1 Cell processing

As ATM cells leave the switch fabric and first enter an output module, they are examined by the cell processing functional block. Cell processing in an output module is very similar to cell processing in an input module. However, instead of filtering out signaling and management cells, an output model must insert such cells back into the outgoing stream of

cells. A functional block diagram of an output module's cell processing block is shown in Figure 39.

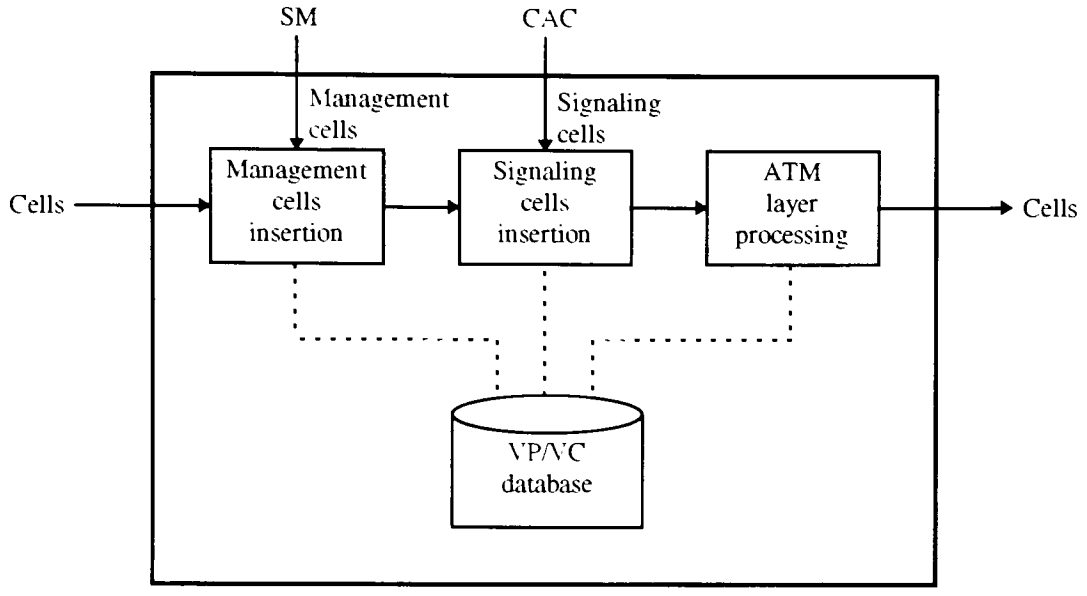


Figure 39 Functional block diagram of cell processing block [1, p. 137]

6.1.1.1 VP/VC database

The VP/VC database located within the cell processing block of an output module contains the information necessary to perform header translation on multicast cells. When a multicast or broadcast cell is encountered by the switch fabric, it is copied the appropriate number of times. Each of these new cells will contain the exact same VPI/VCI values. For this reason, header translation of multicast or broadcast cells cannot be done at the input module. Once the replicated cells have been routed to the correct output ports, the output ports can perform the necessary translation. This ensures that the new VPI/VCI values are assigned after the duplication within the switch fabric has occurred. An internal tag attached to each cell is generally used to indicate to the output module that a cell has been multicast.

The VP/VC database may also be used to store other useful information. For example, the database can store traffic rate statistics for each virtual connection. This data can then be

used by system management to shape the outgoing traffic. The amount of system management information stored in the VP/VC database is determined by the designer of the ATM switch.

6.1.1.2 Signaling cells insertion

The signaling cells insertion block performs the exact opposite of the signaling cells filter found in the input module. The signaling cells insertion block receives new signaling cells that have been generated by the connection admission control portion of the switch and inserts them into the outgoing stream of user data cells. These cells may come directly from a centralized CAC or they may be received by an OM-CAC. The source of the new signaling cells will depend on the distribution of the connection admission control functions in a particular ATM switch.

As discussed earlier, it is also possible for signaling cells to be routed through an ATM switch using the switch fabric. If this is the case, the output module does not need to have any communication with the CAC. Therefore, the signaling cells insertion portion of the cell processing functional block is not required if a switch routes signaling cells through the switch fabric.

6.1.1.3 Management cells insertion

Similar to the signaling cells insertion block, the management cells insertion block receives newly generated management cells and inserts them into the outgoing stream of cells. The management cells insertion block may receive these cells through direct communication with a centralized system management unit, or it may receive them from an OM-SM. It is also possible for system management cells to be routed through the switch fabric. If this is the case, then the management cells insertion block is not required.

6.1.1.4 ATM layer processing

The ATM layer function block is the location where header translation is performed on multicast and broadcasts cells. As discussed earlier, the new VPI/VCI values are stored in the VP/VC database. After passing through the ATM layer processing unit, all outgoing ATM cells will have the proper VPI and VCI values for transmission across the next physical link.

The final step in cell processing is to remove any internal tags that might have been attached to a cell by the input modules. These tags may contain information that can be used to monitor the internal performance of the switch. If so, this data must be processed by the output module. Results can be stored in the VP/VC database, where the system management unit will be able to access them.

6.1.2 OM-CAC and OM-SM

Depending on the design of the overall ATM switch, it is possible that each output module could contain some internal connection admission control and system management functions. The main purpose of an OM-CAC and an OM-SM is to reduce the amount of processing that needs to be performed by the central CAC and SM units. The presence of an OM-CAC and an OM-SM requires that the output module be able to communicate directly with both the central CAC and SM. However, if CAC and SM functions are not distributed to the output modules, then signaling and management cells will be passed to the output modules through the switch fabric.

6.1.3 Transmission convergence

The transmission convergence functional block of an output module performs the physical layer functions that are independent of the physical medium being used. As shown in the functional block diagram of Figure 40, these tasks consist of HEC generation, cell scrambling, and cell mapping and rate decoupling.

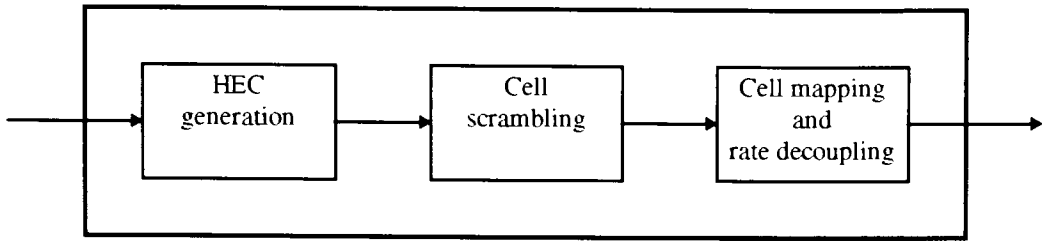


Figure 40 Transmission convergence functional block diagram [1, p. 139]

6.1.3.1 HEC generation

The first job of the transmission convergence functional block is to perform an error check on the header of each cell that it receives. The output module calculates the HEC value for the first four bytes of the header using the same algorithm as the input module. This process is discussed in detail in Section 5.2.1. Once the HEC value has been generated, it is placed in the fifth byte of the cell header. The next input module to receive this cell will confirm this HEC value in order to ensure that no errors occurred while the cell was in transit from one ATM switch to the next.

6.1.3.2 Cell Scrambling

Before leaving the output module, the payload of each ATM cell is scrambled. This is an attempt to prevent payload data from replicating the frame synchronous scrambling sequence used at the section layer. A self-synchronous scrambler with the generator polynomial $1 + x^{43}$ is used for this purpose. Only the 48 payload bytes are scrambled using this algorithm. The five bytes of the header are left untouched. [1, p. 115]

6.1.3.3 Cell mapping and rate decoupling

The final task of the transmission convergence functional block is to place the outgoing ATM cells into SONET frames. This includes the mapping of ATM cells into the payload area of the SONET frames. Within the SONET frame, the sequential order of the

outgoing cells must be preserved. In addition, the byte structure of the cells must be aligned with the byte structure of the SONET frames in which they are placed.

It is also necessary for the output module to perform cell rate decoupling on the outgoing data stream. Due to the synchronous nature of the SONET protocol, an ATM cell must be transmitted during every available time slot. If the output module does not have a cell to send, it must insert idle cells into the cell stream. These unassigned cells are given a value of zero for both the VPI and VCI fields of the header. In this manner, a continuous stream of cells is available to fill the payload of each SONET frame.

6.1.4 SONET functions

The SONET functional block is the last unit that ATM cells pass through before they leave the switch. This portion of the output module is responsible for handling the physical level considerations associated with transmitting data over optical fiber according to the SONET protocol. Figure 41 illustrates a function block diagram of this portion of an output module.

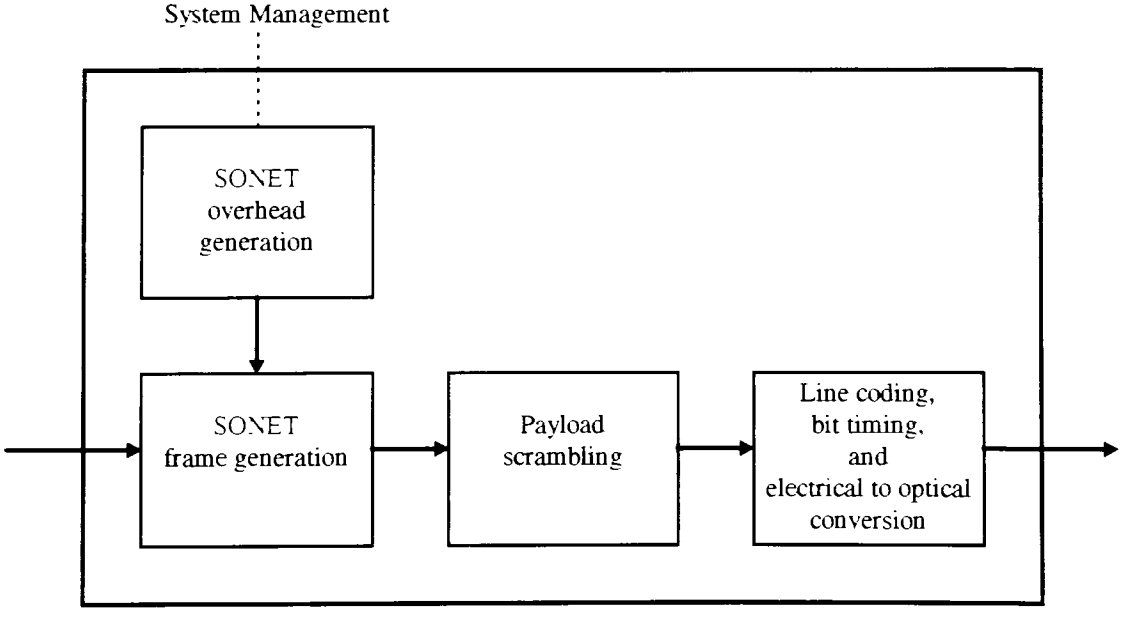


Figure 41 SONET functional block of an output module [1, p. 140]

The SONET functional block is necessary in order to correctly generate each of the SONET frames for transmission along the physical link. The functions performed by this portion of the output module are dependent on the transport protocol being used to carry cells from one ATM switch to the next. If SONET was not the chosen protocol, then this functional block would have to be replaced with one that complied with the protocol in use. The SONET functions and the generation of a SONET frame are covered in detail by Chapter 3.

6.2 C++ Model

The C++ model of an output module is very similar to the one provided for an input module. In order to provide a functional reference for the VHDL model, the software model of an output module performs many of the same functions of the input model, only in reverse. This allowed for the reuse of the "ATMCell" and "SONETFrame" classes and many of their associated member functions. The implementation of these classes are discussed in Sections 5.2.1 and 5.2.2.

6.2.1 output.cc

The file "output.cc" contains the top level C++ code for the object oriented model of an ATM output module designed to interface with a SONET transport network. An executable version of this code can be created by running the accompanying "makefile." When the executable program is started, it first prompt the user whether it should create a new input test file. If a positive response is indicated, the program will ask for the number of random ATM cells to be generated. It then asks for the percentage of these cells that should be valid. The new cells are stored in the file "outmodcells.in." If the user does not choose to generate a new input file, then the program reads the stream of ATM cells from the existing "outmodcells.in" file. Once each cell has its header checked for errors, the cells are mapped into SONET frames. As these frames are output by the output module, they are written to the file "outmodframes.out".

In order for a user to simulate the functionality of an output module using this model, the desired stream of input cells must be stored in sequential order in the file "outmodcells.in" Once the executable version of the model has been run, all other tasks will be handled by the software automatically. The file "outmodframes.out" can then be checked for correctness.

6.3 VHDL Model

In addition to providing a C++ software model, another purpose of this thesis was to design and implement a VHDL functional model of an output module. This model can be used as a tool to study the performance effects of different ATM switch architectures in a simulation environment. The correct functionality of this VHDL model has been verified using the C++ model discussed in the previous section.

The operation of an output module is similar to that of an input module, except that it must perform many of the same operations in reverse. For this reason, many of the VHDL design files are common to both the input module and output module implementations. The output module consists of six VHDL files. They are "inoutmod_pb.vhd", "xor_ea.vhd", "shift_ea.vhd", "crc_check_ea.vhd", "output_module_ea.vhd", and "output_module_tb.vhd". The VHDL code in each of these files follows the same conventions used by the input module as discussed in Section 5.3. In addition, Section 5.3 includes a detailed discussion of all of the VHDL files that are common to both designs. Only the files that are unique to the output module model will be discussed here.

6.3.1 output_module_ea.vhd

This file contains the entity declaration and the behavioral architecture for an output module used to interface between an ATM switch and a SONET transport network. An external view of this entity is shown in Figure 42.

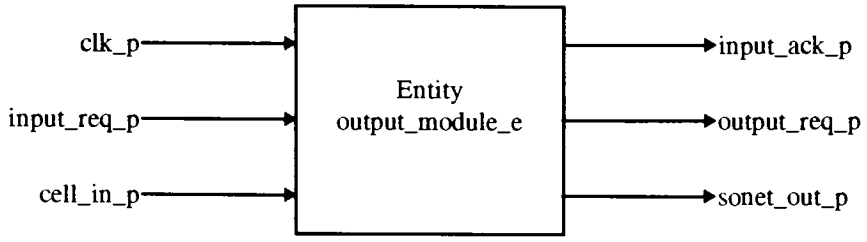


Figure 42 Output module symbol

The 6.43 nsec clock used by the SONET transport network must be provided to the output module through signal port "clk_p". When a valid cell is available for input through the port "cell_in_p" the entity transmitting that cell must assert the "input_req_p" line. When the output module recognizes that a new cell is ready to be input, it reads from "cell_in_p" and pulses the "input_ack_p" line to indicate that the new cell has been successfully read. Each time a rising edge of the "clk_p" signal occurs and the "output_req_p" signal is asserted, the output module places a single bit of a SONET frame onto the "sonet_out_p" port.

The job of the output module is slightly different than that of the input module in regards to each cell's header control byte. While the input module is responsible for checking the validity of each HEC byte, the output module must generate this byte before a cell is transmitted. The same divisor circuit can be used to perform both tasks. In order to generate the HEC byte, the output module must shift the first 32 bits of each header into the "crc_check_e" divisor circuit. Following these four bytes, another byte containing all zeros must be shifted in. These eight zero bits represent the HEC byte. Once all 40 bits have been shifted in, the byte stored in the register will be the remainder produced by the division. This value is also the HEC byte, and must be placed into the fifth byte of the cell header before the cell is transmitted. A complete example of this process is given in Appendix C. The exact difference between the way in which header error control calculations are performed by the output module and input module can be examined by comparing the example in Appendix C with that in Appendix A.

6.3.2 output_module_tb.vhd

The file "output_module_tb.vhd" contains a testbench entity and architecture that can be used to test and monitor the operation of the output module. This entity instantiates a single "output_module_e" component and also generates a SONET clock signal to drive it. The testbench uses a behavioral architecture consisting of two processes to send test data to and receive results from the output module.

The first process in this testbench is responsible for reading ATM cells from and ASCII file and sending them to the output module that is being tested. The ATM cells being used for the test must be stored in a file named "outmodcells.in". The other process within the "output_module_tb.vhd" testbench receives SONET frames as they are transmitted by the output module and writes them to an ASCII file named "outmodframes.out". The use of ASCII files to store input data and output results allow identical test cases to be executed with both the C++ and VHDL models. The test results can then be compared to verify that both models are functioning correctly.

6.4 Testing procedure

A test case can be conducted by first placing the desired input data into the file "outmodcells.in". The entity "output_module_tb" and the architecture "testbench_a" must then be loaded into the Mentor Graphics QuickVHDL simulator. The simulation must be run long enough for the entire input file to be processed and output in SONET frame format. The output file "outmodframes.out" can then be examined. An example test case has been documented in Appendix D.

The types of test cases used to verify the functionality of the output module were similar to those used to test the input module. Different volumes and patterns of ATM traffic were applied to the output module. In each case, the output of both the C++ model and the VHDL model were identical. This confirmed the proper functionality of both models.

7 Conclusions

ATM and SONET have emerged as leading technologies in the attempt to integrate different types of data traffic over a single, high-speed digital communications network. For this reason, it is important to study the interface between these two separate protocols. This thesis has attempted to do so by first examining the implementation of each of these technologies in detail. The main focus was then placed on how a complete ATM switching system can be designed using SONET as the underlying transport network. In particular, the input and output modules used to interface an ATM switch with the SONET network were studied. Finally, C++ and VHDL functional models of both an input and an output module were developed and tested.

As ATM technology has been becoming more popular, much emphasis has been placed on the design of the ATM switch fabric and its ability to route and buffer cells. However, this thesis has demonstrated that a complete ATM switch is responsible for many other important tasks that are necessary to maintain a properly functioning data communications network. One possible scheme for partitioning these tasks into separate functional blocks has been examined. In addition to the switch fabric, an ATM switch must also contain input and output modules, a connection admission control unit, and a system management unit.

The proper design and interconnection of each of these functional blocks is critical to the performance of an ATM switching system. The design and operation of the entire ATM switch depends on the degree to which each of the critical functions, such as CAC and SM, is distributed throughout the switch. Although a high degree of distribution can avoid a possible bottleneck created by centralized processing, it will also increase the complexity of each of the switch's functional components. The partitioning of tasks and the degree of distribution within an ATM switch is an important yet complicated issue that has a direct effect on overall network performance. This thesis has introduced the issues and provided the tools for future study of this topic.

The main focus of this thesis has been the input and output modules of an ATM switch that interfaces specifically with a SONET network. The input module is especially important because all traffic entering an ATM switch must first pass through an input module. It is responsible for terminating the SONET signal, extracting ATM cells from each SONET frame, and preparing these cells to be routed through the switch fabric. Although the output module is similar in operation, it is slightly less complicated than the input module. An output module receives ATM cells from the switch fabric and prepares them for delivery to the next node in the network by inserting them into SONET frames. This thesis has examined the function and design of both input modules and output modules in detail.

In addition to discussing design issues associated with input and output modules, a C++ and VHDL model of each has also been provided. The purpose of these models is to provide a tool to be used in the further study of the performance issues related to the architecture of an overall ATM switch. The input and output modules are only two of the five main functional blocks contained within an ATM switch. If models of the other functional blocks were developed, they could be integrated with the input and output modules provided here to form a complete ATM switch (see [14] for the model of a switch fabric). Having separate models for each of the functional components of a switch allows them to be easily interconnected in many configurations. The performance of several of the possible architectures discussed in this thesis could then be investigated.

Furthermore, the models developed for this thesis specify only the functional requirements of an input and output module. Only the required features of each component have been implemented by these models. The behavioral nature of these models allows them to be easily altered, as they are not dependent on a specific implementation. By adding the necessary functionality, these models can be utilized to study the effects of task distribution on overall switch performance. The C++ models have been created to verify that each of the VHDL models is actually producing the expected results for a particular test case. The inclusion of testbenchs that read input from and write output to ASCII files helps to automate the testing procedure and allows specific test cases to be easily generated and reproduced.

7.1 Problems encountered

During the course of this thesis, only a few minor problems were encountered. One of the most significant was the difficulty in obtaining information that provided a detailed explanation of the functionality of both input and output modules for an ATM switch. There is an abundance of research available that focused on the design of an ATM switch fabric, but relatively little information pertaining to the other parts of a switch. This is one of the main reasons that this thesis was proposed. The study of overall switch design is definitely an important topic that has plenty of opportunity for further research. Since the switch fabric is the core of an ATM switch, most research has been focused on that one functional block. This makes it difficult to find information related to the other parts of an ATM switch.

Another minor problem encountered was the amount of time required to run a VHDL simulation for large test cases. The simulation of test cases in which tens of thousands of cells were passed through the input or output module often took several hours to complete. However, this delay was not unexpected given the complexity of the large test cases. The speed of the simulator was mostly an inconvenience and not a major problem. The simulator only took a few minutes to run the smaller test cases.

7.2 Suggestions for improvement

If this thesis was to be repeated, several aspects might be changed or improved. Most of these changes would affect the implementation of the C++ and VHDL models. The first of these might be the manner in which test data was stored in ASCII files. Although this method of placing all input and output data into separate ASCII files is convenient and effective, large test cases require several megabytes of file space. Although being able to examine the exact results of each test is helpful, it may be more efficient to further automate the testing procedure by having the VHDL model and C++ model automatically compare results and only log any differences.

7.3 Future study

The work completed in this thesis can be used as the foundation for further study in the area of ATM switch design. This thesis has examined several different design strategies for constructing a complete ATM switch. The performance and implementation issues associated with each overall architecture have also been studied. Special focus has been placed on the functionality of the input and output modules. The next step would be to actually implement several of theses different designs and compare the performance of each one. The models of input and output modules provided by this thesis can be used to construct these switch architectures. They can be integrated with models of a switch fabric, connection admission control, and system management units in order to create a complete ATM switch. The models provided here are flexible enough to allow simple modifications and expansions. This type of research will be beneficial in the development of faster and more efficient ATM switches.

References

[1] Thomas M. Chen & Stephens S. Liu, "ATM Switching Systems." Artech House, Inc., Boston, 1995.

[2] Harry J.R. Dutton & Peter Lenhard, "Asynchronous Transfer Mode (ATM): Technical Overview." Prentice-Hall, Inc., New Jersey, 1995.

[3] Uyless Black, "ATM: Foundation for Broadband Networks." Prentice-Hall, Inc., New Jersey, 1995.

[4] James Martin, "Asynchronous Transfer Mode: ATM Architecture and Implementation." Prentice-Hall, Inc., New Jersey, 1997.

[5] David E. McDysan & Darren L. Spohn, "ATM: Theory and Application." McGraw-Hill, Inc., New York, 1995.

[6] John E. Midwinter, "Photonics in Switching: Background and Components." Academic Press, Inc., Boston, 1993.

[7] Gary C. Kessler & Peter Southwick, "ISDN: Concepts, Facilities, and Services." McGraw-Hill, Inc., New York, 1995.

[8] Raif O. Onvural, "Asynchronous Transfer Mode Networks: Performance Issues." Artech House, Inc., 1995.

[9] J.M. Pitts & J.A. Schormans, "Introduction to ATM Design and Performance." John Wiley & Sons, New York, 1996.

[10] Darren L. Spohn, "Data Network Design." McGraw-Hill, Inc., New York, 1993.

[11] The ATM Forum, "ATM User-Network Interface Specifications." Prentice-Hall, Inc., New Jersey, 1993.

[12] Ronald J. Vetter, "ATM Concepts, Architectures, and Protocols," *Communications of the ACM*, Vol. 38, February 1995, pp. 30-38.

[13] Jeffery L. Krieger, "A Performance Evaluation of Several ATM Switching Architectures." Master's thesis, Rochester Institute of Technology, January 1996.

[14] Rudi Rughoonundon, "The Design of a Single Chip 8x8 ATM Switch in 0.5 μm CMOS VLSI." Master's thesis, Rochester Institute of Technology, January 1996.

[15] Uyless Black & Sharleen Waters, "SONET and T1: Architectures for Digital Transport Networks." Prentice Hall, Inc., New Jersey, 1997.

[16] William Stallings, "Data and Computer Communications." Macmillan Publishing Company, New York, 1994.

[17] Andrew Tanenbaum, "Computer Networks." Prentice Hall, Inc., New Jersey, 1996.

Appendix A

The following table contains a step by step example of the manner in which the header error control calculation is performed by the input module. The hexadecimal representation of the error free header used in this example is "01 02 03 04 E3". After 40 shifts, the register contains all zeros, thus indicating that the HEC value is a valid one.

	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	$C_7 \oplus C_1$	$C_7 \oplus C_0$	$C_7 \oplus In$	In
Initial	0	0	0	0	0	0	0	0	0	0	0	0
Step 1	0	0	0	0	0	0	0	0	0	0	0	0
Step 2	0	0	0	0	0	0	0	0	0	0	0	0
Step 3	0	0	0	0	0	0	0	0	0	0	0	0
Step 4	0	0	0	0	0	0	0	0	0	0	0	0
Step 5	0	0	0	0	0	0	0	0	0	0	0	0
Step 6	0	0	0	0	0	0	0	0	0	0	0	0
Step 7	0	0	0	0	0	0	0	0	0	0	1	1
Step 8	0	0	0	0	0	0	0	1	0	1	0	0
Step 9	0	0	0	0	0	0	1	0	1	0	0	0
Step 10	0	0	0	0	0	1	0	0	0	0	0	0
Step 11	0	0	0	0	1	0	0	0	0	0	0	0
Step 12	0	0	0	1	0	0	0	0	0	0	0	0
Step 13	0	0	1	0	0	0	0	0	0	0	0	0
Step 14	0	1	0	0	0	0	0	0	0	0	1	1
Step 15	1	0	0	0	0	0	0	1	1	0	1	0
Step 16	0	0	0	0	0	1	0	1	0	1	0	0
Step 17	0	0	0	0	1	0	1	0	1	0	0	0
Step 18	0	0	0	1	0	1	0	0	0	0	0	0
Step 19	0	0	1	0	1	0	0	0	0	0	0	0
Step 20	0	1	0	1	0	0	0	0	0	0	0	0
Step 21	1	0	1	0	0	0	0	0	1	1	1	0

Step 22	0	1	0	0	0	1	1	1	1	1	1	1
Step 23	1	0	0	0	1	1	1	1	0	0	0	1
Step 24	0	0	0	1	1	0	0	0	0	0	0	0
Step 25	0	0	1	1	0	0	0	0	0	0	0	0
Step 26	0	1	1	0	0	0	0	0	0	0	0	0
Step 27	1	1	0	0	0	0	0	0	1	1	1	0
Step 28	1	0	0	0	0	1	1	1	0	0	1	0
Step 29	0	0	0	0	1	0	0	1	0	1	1	1
Step 30	0	0	0	1	0	0	1	1	1	1	0	0
Step 31	0	0	1	0	0	1	1	0	1	0	0	0
Step 32	0	1	0	0	1	1	0	0	0	0	1	1
Step 33	1	0	0	1	1	0	0	1	1	0	0	1
Step 34	0	0	1	1	0	1	0	0	0	0	1	1
Step 35	0	1	1	0	1	0	0	1	0	1	0	0
Step 36	1	1	0	1	0	0	1	0	0	1	1	0
Step 37	1	0	1	0	0	0	1	1	0	0	1	0
Step 38	0	1	0	0	0	0	0	1	0	1	1	1
Step 39	1	0	0	0	0	0	1	1	0	0	0	1
Step 40	0	0	0	0	0	0	0	0				

Appendix B

This appendix includes the results of a simple test case that was run on the input module. The "inmodframes.in" file shown here was used as input to both the C++ and the VHDL models. The file contains a single SONET frame which holds 40 valid ATM cells followed by 4 idle cells. Each row of the frame is grouped together, and consecutive rows are separated by a blank line. Each byte is represented in decimal format.

The C++ and the VHDL model each produced a file named "inmodcells.out". The contents of these files were identical and are included in this appendix. The file "inmodcells.out" lists each of the ATM cells transmitted by the input module, each separated by a blank line. All of the 40 valid cells are present within this file, and each of the idle cells has been removed from the incoming bit stream.

A sample waveform from the VHDL simulation of the entity "input_module_tb.vhd" is also included. This simulation used the file "inmodframes.in" as input. The time frame captured in this waveform shows the first cell being received by the input module. The header of the cell is shifted into the "crc_check_e" entity until a remainder of zero is encountered. At that time, the input module receives that cell's payload.

0 0 0 0 0 0 0 0 0 0 178 26 221 70 23 22 176 195 9 108
27 74 25 162 169 161 205 182 11 11 80 108 142 51 118 104 79 51 3 52
66 207 219 204 117 241 251 1 197 15 45 87 97 134 212 186 210 164 231 39
122 212 65 229 133 17 209 154 172 176 220 214 34 213 42 199 59 87 175 5
188 209 229 169 234 61 191 43 238 126 115 123 137 61 98 175 6 183 222 104
183 198 12 88 249 48 149 168 225 245 244 19 174 225 137 106 152 111 196 93
73 194 49 117 36 214 144 137 118 83 6 60 188 193 152 191 129 232 235 75
95 246 174 52 115 95 183 243 5 190 249 203 85 109 199 136 4 29 126 36
124 137 25 195 189 181 201 188 146 203 218 246 232 116 89 178 142 241 139 75
104 36 236 51 201 244 71 94 26 217 102 154 86 147 254 95 244 107 181 178
7 216 119 188 185 193 36 200 5 48 191 78 52 80 48 189 19 232 189 49
113 187 127 69 170 116 119 238 178 40 63 64 132 199 210 5 225 214 171 77
37 243 51 228 74 225 71 7 142 180 100 11 45 153 173 176 254 166 46 217
201 129 219 226 156 194 36 215 224 99

0 0 0 0 0 0 0 0 0 0 148 68 152 37 227 177 48 221 0 60
6 51 64 13 244 63 167 1 29 144 99 227 210 235 78 11 226 248 109 123
142 190 117 92 225 40 44 129 105 65 147 27 144 202 126 8 133 234 209 247
254 4 50 191 75 128 217 13 101 154 145 138 126 155 180 217 90 170 249 6
175 182 62 240 154 215 178 40 100 96 167 248 175 150 238 54 212 55 35 189
214 162 131 129 135 70 76 250 179 168 186 225 203 166 167 129 106 210 231 14
53 151 133 195 22 79 178 180 241 40 95 180 229 94 206 236 253 210 220 121
130 60 222 86 133 99 158 30 246 204 140 157 80 170 91 69 110 243 252 77
119 223 75 9 112 159 77 202 209 149 217 206 193 94 75 239 119 34 191 72
53 139 117 147 110 68 12 230 27 139 138 99 64 92 148 220 5 144 151 38
78 54 197 99 152 226 127 21 135 91 224 179 78 244 138 110 87 127 115 244
238 160 65 224 55 118 134 126 218 170 173 142 222 181 164 195 72 41 163 186
128 200 24 193 104 6 55 109 90 180 28 203 46 125 246 189 184 147 116 83
78 201 73 77 105 79 113 182 145 39

0 0 0 0 0 0 0 0 0 0 15 71 156 146 239 108 104 30 169 174
49 69 93 57 179 244 182 61 145 253 227 131 105 152 119 165 249 237 204 144
197 89 99 167 127 54 173 183 182 15 14 7 168 7 140 54 185 32 80 110
98 52 117 30 155 195 181 64 151 111 133 94 215 100 48 17 196 252 139 55
86 222 178 158 225 24 50 38 178 176 242 147 107 31 214 18 51 179 62 221
120 46 228 9 153 124 64 206 36 166 213 81 234 142 87 196 45 111 107 216
15 66 119 109 19 0 23 154 68 80 147 33 183 238 172 103 88 193 198 74
211 64 232 248 90 197 246 84 192 190 103 196 13 57 39 58 4 172 81 248
17 144 19 140 82 103 58 249 147 188 67 18 124 94 110 253 212 70 41 74
133 246 218 174 56 184 8 249 28 146 32 116 98 2 204 160 126 33 159 79
83 111 75 155 204 61 248 31 144 126 62 37 50 83 125 127 66 83 227 102
29 225 203 23 185 160 46 25 5 187 250 81 124 239 42 102 74 205 57 161
199 209 77 0 35 251 175 218 215 131 231 249 40 208 31 86 207 212 109 57
23 102 120 208 122 234 21 107 108 248

0 0 0 0 0 0 0 0 0 0 84 224 18 191 177 90 234 97 195 125
68 13 36 174 122 48 218 221 241 139 228 185 185 21 134 8 95 209 20 94
103 143 10 173 211 128 78 77 225 131 220 156 249 16 187 115 99 78 164 87
88 216 174 85 93 62 226 0 105 190 136 103 204 238 147 139 226 162 99 69
17 125 31 122 186 83 226 59 193 62 128 44 13 244 198 15 116 174 88 77
251 109 183 100 77 186 135 146 237 244 35 221 153 69 69 101 86 217 232 6
78 121 146 47 138 156 27 154 114 151 203 22 176 195 9 20 17 59 107 48
111 124 157 46 197 186 2 126 20 59 8 177 202 170 248 235 108 124 122 128
231 232 135 246 150 220 51 50 27 87 87 21 167 30 8 102 214 173 235 197
34 218 75 243 73 154 246 148 156 39 187 5 253 36 55 252 92 221 232 142
181 153 2 226 10 200 89 132 159 116 172 59 120 62 227 26 75 207 158 32
158 209 216 211 192 202 108 23 58 126 81 60 240 41 193 0 29 251 6 170
31 51 189 187 30 76 217 241 219 11 245 151 28 109 159 200 60 226 71 8
17 211 124 220 251 113 20 94 172 32

0 0 0 0 0 0 0 0 0 0 136 224 14 11 135 217 208 236 109 187
39 14 165 117 11 155 49 41 51 69 47 239 220 43 2 83 61 182 26 142
171 59 185 143 69 168 165 164 142 23 59 196 183 28 8 4 12 60 105 105
89 62 243 65 35 250 247 62 218 77 152 144 138 181 225 77 25 90 199 183
17 231 110 226 91 144 182 106 106 189 207 103 67 102 26 104 167 243 182 92
42 85 112 19 194 102 167 187 12 60 252 98 191 84 81 200 122 101 231 100
244 219 201 196 51 216 196 181 243 60 170 114 190 120 156 150 21 69 195 66
116 0 189 104 53 247 33 121 22 65 162 67 148 47 28 75 108 65 21 51
240 19 107 188 217 149 206 227 29 51 39 213 208 170 220 70 112 86 40 84
137 155 95 121 157 198 95 207 191 200 176 118 205 65 23 63 129 130 173 91
89 196 89 207 208 75 181 226 37 68 190 195 190 37 27 155 125 210 235 201
117 241 109 105 194 61 97 221 67 156 121 246 99 93 67 3 200 72 77 198
132 231 250 154 169 89 156 172 191 27 207 98 21 183 115 113 126 221 131 186
169 127 157 188 114 74 216 162 203 62

0 0 0 0 0 0 0 0 0 0 202 65 196 8 197 38 171 175 42 23
222 191 71 28 69 114 49 106 39 25 115 84 174 56 17 174 6 93 227 4
67 191 51 157 23 207 23 34 11 44 107 59 191 112 89 245 59 34 86 13
170 250 254 194 23 63 154 27 95 238 29 150 205 222 60 115 63 21 2 133
18 74 231 31 219 22 108 57 224 240 144 111 227 89 64 204 234 203 138 245
207 35 218 247 114 69 54 179 205 184 208 225 65 178 43 62 187 109 58 45
238 24 176 10 3 131 223 185 31 225 99 17 89 184 237 55 74 226 125 33
98 148 91 67 250 74 238 193 228 184 0 69 85 126 59 137 6 170 75 252
141 39 182 127 109 244 58 91 254 51 194 53 190 195 208 242 247 239 58 224
54 75 5 177 159 45 96 17 7 117 234 101 212 138 199 70 118 134 116 133
193 126 6 181 254 53 45 212 182 146 97 208 195 231 254 53 46 248 41 66
138 191 44 152 175 81 151 10 53 228 92 239 225 51 246 167 129 175 1 8
52 86 32 113 204 9 179 249 176 42 18 29 155 66 151 212 106 185 147 219
236 101 32 153 213 138 193 87 51 220

0 0 0 0 0 0 0 0 0 0 132 93 74 59 128 36 238 170 227 236
194 119 209 200 166 226 175 35 202 207 190 212 47 216 165 76 120 179 188 219
173 28 220 157 197 50 195 48 34 31 116 115 138 5 20 172 81 219 81 156

48 176 140 16 17 82 106 65 29 196 165 246 192 252 57 191 181 92 100 44
206 109 31 77 168 213 251 117 246 128 28 113 4 223 6 16 40 237 72 68
73 239 211 46 210 183 158 193 185 136 97 152 245 136 47 59 28 69 217 154
1 21 27 32 136 149 181 123 172 34 19 101 168 250 144 237 92 124 125 31
196 34 7 3 165 45 147 113 195 52 62 222 68 225 61 137 104 177 102 182
112 71 64 102 94 208 97 45 233 167 121 225 202 65 141 112 27 27 105 110
190 153 142 200 102 165 165 179 60 235 80 37 194 20 204 14 106 89 85 198
91 245 191 119 116 196 250 11 62 35 17 21 80 157 211 119 235 37 42 185
170 152 51 45 232 249 2 133 70 22 173 190 167 1 135 123 127 103 34 209
86 139 250 36 76 229 227 248 149 82 14 126 121 198 169 75 55 139 70 148
88 242 221 111 195 104 76 225 47 241

0 0 0 0 0 0 0 0 0 0 235 5 18 41 71 190 97 240 42 194
126 156 144 63 146 96 194 116 231 53 104 158 125 121 90 10 247 243 10 51
163 156 185 193 166 172 176 220 214 151 33 70 172 38 167 151 79 114 128 250
110 102 246 178 95 234 39 246 74 149 120 90 130 118 241 139 209 103 153 5
98 84 235 105 28 169 28 168 153 203 154 173 130 206 164 160 245 207 95 28
144 98 190 183 198 70 115 220 82 174 32 26 168 251 30 124 121 57 67 164
165 214 73 128 137 67 110 200 134 100 59 82 71 79 65 162 106 152 85 63
120 168 159 66 208 82 121 43 9 169 247 146 6 194 236 74 205 72 94 192
145 13 141 206 50 87 136 86 129 63 20 28 163 84 225 125 136 185 4 230
30 1 246 144 140 9 60 173 88 107 149 129 225 198 183 115 233 243 161 165
50 112 94 33 69 113 118 120 106 100 72 247 142 70 199 236 125 74 133 150
13 135 6 109 116 33 3 109 177 144 232 68 222 115 251 136 100 69 75 169
26 15 108 18 45 5 21 71 71 17 151 113 80 58 87 146 73 120 91 169
4 243 122 200 17 219 250 129 203 44

0 0 0 0 0 0 0 0 0 0 227 51 35 69 13 204 72 76 158 177
248 101 24 25 5 215 74 246 155 56 253 141 150 7 196 109 142 89 40 228
86 27 183 186 192 123 192 191 70 114 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

178 26 221 70 23 22 176 195 9 108
27 74 25 162 169 161 205 182 11 11
80 108 142 51 118 104 79 51 3 52
66 207 219 204 117 241 251 1 197 15
45 87 97 134 212 186 210 164 231 39
122 212 65

229 133 17 209 154 172 176 220 214 34
213 42 199 59 87 175 5 188 209 229
169 234 61 191 43 238 126 115 123 137
61 98 175 6 183 222 104 183 198 12
88 249 48 149 168 225 245 244 19 174
225 137 106

152 111 196 93 73 194 49 117 36 214
144 137 118 83 6 60 188 193 152 191
129 232 235 75 95 246 174 52 115 95
183 243 5 190 249 203 85 109 199 136
4 29 126 36 124 137 25 195 189 181
201 188 146

203 218 246 232 116 89 178 142 241 139
75 104 36 236 51 201 244 71 94 26
217 102 154 86 147 254 95 244 107 181
178 7 216 119 188 185 193 36 200 5
48 191 78 52 80 48 189 19 232 189
49 113 187

127 69 170 116 119 238 178 40 63 64
132 199 210 5 225 214 171 77 37 243
51 228 74 225 71 7 142 180 100 11
45 153 173 176 254 166 46 217 201 129
219 226 156 194 36 215 224 99 148 68
152 37 227

177 48 221 0 60 6 51 64 13 244
63 167 1 29 144 99 227 210 235 78
11 226 248 109 123 142 190 117 92 225
40 44 129 105 65 147 27 144 202 126
8 133 234 209 247 254 4 50 191 75
128 217 13

101 154 145 138 126 155 180 217 90 170
249 6 175 182 62 240 154 215 178 40
100 96 167 248 175 150 238 54 212 55
35 189 214 162 131 129 135 70 76 250

179 168 186 225 203 166 167 129 106 210
231 14 53

151 133 195 22 79 178 180 241 40 95
180 229 94 206 236 253 210 220 121 130
60 222 86 133 99 158 30 246 204 140
157 80 170 91 69 110 243 252 77 119
223 75 9 112 159 77 202 209 149 217
206 193 94

75 239 119 34 191 72 53 139 117 147
110 68 12 230 27 139 138 99 64 92
148 220 5 144 151 38 78 54 197 99
152 226 127 21 135 91 224 179 78 244
138 110 87 127 115 244 238 160 65 224
55 118 134

126 218 170 173 142 222 181 164 195 72
41 163 186 128 200 24 193 104 6 55
109 90 180 28 203 46 125 246 189 184
147 116 83 78 201 73 77 105 79 113
182 145 39 15 71 156 146 239 108 104
30 169 174

49 69 93 57 179 244 182 61 145 253
227 131 105 152 119 165 249 237 204 144
197 89 99 167 127 54 173 183 182 15
14 7 168 7 140 54 185 32 80 110
98 52 117 30 155 195 181 64 151 111
133 94 215

100 48 17 196 252 139 55 86 222 178
158 225 24 50 38 178 176 242 147 107
31 214 18 51 179 62 221 120 46 228
9 153 124 64 206 36 166 213 81 234
142 87 196 45 111 107 216 15 66 119
109 19 0

23 154 68 80 147 33 183 238 172 103
88 193 198 74 211 64 232 248 90 197
246 84 192 190 103 196 13 57 39 58
4 172 81 248 17 144 19 140 82 103
58 249 147 188 67 18 124 94 110 253
212 70 41

74 133 246 218 174 56 184 8 249 28
146 32 116 98 2 204 160 126 33 159
79 83 111 75 155 204 61 248 31 144

126 62 37 50 83 125 127 66 83 227
102 29 225 203 23 185 160 46 25 5
187 250 81

124 239 42 102 74 205 57 161 199 209
77 0 35 251 175 218 215 131 231 249
40 208 31 86 207 212 109 57 23 102
120 208 122 234 21 107 108 248 84 224
18 191 177 90 234 97 195 125 68 13
36 174 122

48 218 221 241 139 228 185 185 21 134
8 95 209 20 94 103 143 10 173 211
128 78 77 225 131 220 156 249 16 187
115 99 78 164 87 88 216 174 85 93
62 226 0 105 190 136 103 204 238 147
139 226 162

99 69 17 125 31 122 186 83 226 59
193 62 128 44 13 244 198 15 116 174
88 77 251 109 183 100 77 186 135 146
237 244 35 221 153 69 69 101 86 217
232 6 78 121 146 47 138 156 27 154
114 151 203

22 176 195 9 20 17 59 107 48 111
124 157 46 197 186 2 126 20 59 8
177 202 170 248 235 108 124 122 128 231
232 135 246 150 220 51 50 27 87 87
21 167 30 8 102 214 173 235 197 34
218 75 243

73 154 246 148 156 39 187 5 253 36
55 252 92 221 232 142 181 153 2 226
10 200 89 132 159 116 172 59 120 62
227 26 75 207 158 32 158 209 216 211
192 202 108 23 58 126 81 60 240 41
193 0 29

251 6 170 31 51 189 187 30 76 217
241 219 11 245 151 28 109 159 200 60
226 71 8 17 211 124 220 251 113 20
94 172 32 136 224 14 11 135 217 208
236 109 187 39 14 165 117 11 155 49
41 51 69

47 239 220 43 2 83 61 182 26 142
171 59 185 143 69 168 165 164 142 23

59 196 183 28 8 4 12 60 105 105
89 62 243 65 35 250 247 62 218 77
152 144 138 181 225 77 25 90 199 183
17 231 110

226 91 144 182 106 106 189 207 103 67
102 26 104 167 243 182 92 42 85 112
19 194 102 167 187 12 60 252 98 191
84 81 200 122 101 231 100 244 219 201
196 51 216 196 181 243 60 170 114 190
120 156 150

21 69 195 66 116 0 189 104 53 247
33 121 22 65 162 67 148 47 28 75
108 65 21 51 240 19 107 188 217 149
206 227 29 51 39 213 208 170 220 70
112 86 40 84 137 155 95 121 157 198
95 207 191

200 176 118 205 65 23 63 129 130 173
91 89 196 89 207 208 75 181 226 37
68 190 195 190 37 27 155 125 210 235
201 117 241 109 105 194 61 97 221 67
156 121 246 99 93 67 3 200 72 77
198 132 231

250 154 169 89 156 172 191 27 207 98
21 183 115 113 126 221 131 186 169 127
157 188 114 74 216 162 203 62 202 65
196 8 197 38 171 175 42 23 222 191
71 28 69 114 49 106 39 25 115 84
174 56 17

174 6 93 227 4 67 191 51 157 23
207 23 34 11 44 107 59 191 112 89
245 59 34 86 13 170 250 254 194 23
63 154 27 95 238 29 150 205 222 60
115 63 21 2 133 18 74 231 31 219
22 108 57

224 240 144 111 227 89 64 204 234 203
138 245 207 35 218 247 114 69 54 179
205 184 208 225 65 178 43 62 187 109
58 45 238 24 176 10 3 131 223 185
31 225 99 17 89 184 237 55 74 226
125 33 98

148 91 67 250 74 238 193 228 184 0

69 85 126 59 137 6 170 75 252 141
39 182 127 109 244 58 91 254 51 194
53 190 195 208 242 247 239 58 224 54
75 5 177 159 45 96 17 7 117 234
101 212 138

199 70 118 134 116 133 193 126 6 181
254 53 45 212 182 146 97 208 195 231
254 53 46 248 41 66 138 191 44 152
175 81 151 10 53 228 92 239 225 51
246 167 129 175 1 8 52 86 32 113
204 9 179

249 176 42 18 29 155 66 151 212 106
185 147 219 236 101 32 153 213 138 193
87 51 220 132 93 74 59 128 36 238
170 227 236 194 119 209 200 166 226 175
35 202 207 190 212 47 216 165 76 120
179 188 219

173 28 220 157 197 50 195 48 34 31
116 115 138 5 20 172 81 219 81 156
48 176 140 16 17 82 106 65 29 196
165 246 192 252 57 191 181 92 100 44
206 109 31 77 168 213 251 117 246 128
28 113 4

223 6 16 40 237 72 68 73 239 211
46 210 183 158 193 185 136 97 152 245
136 47 59 28 69 217 154 1 21 27
32 136 149 181 123 172 34 19 101 168
250 144 237 92 124 125 31 196 34 7
3 165 45

147 113 195 52 62 222 68 225 61 137
104 177 102 182 112 71 64 102 94 208
97 45 233 167 121 225 202 65 141 112
27 27 105 110 190 153 142 200 102 165
165 179 60 235 80 37 194 20 204 14
106 89 85

198 91 245 191 119 116 196 250 11 62
35 17 21 80 157 211 119 235 37 42
185 170 152 51 45 232 249 2 133 70
22 173 190 167 1 135 123 127 103 34
209 86 139 250 36 76 229 227 248 149
82 14 126

121 198 169 75 55 139 70 148 88 242
221 111 195 104 76 225 47 241 235 5
18 41 71 190 97 240 42 194 126 156
144 63 146 96 194 116 231 53 104 158
125 121 90 10 247 243 10 51 163 156
185 193 166

172 176 220 214 151 33 70 172 38 167
151 79 114 128 250 110 102 246 178 95
234 39 246 74 149 120 90 130 118 241
139 209 103 153 5 98 84 235 105 28
169 28 168 153 203 154 173 130 206 164
160 245 207

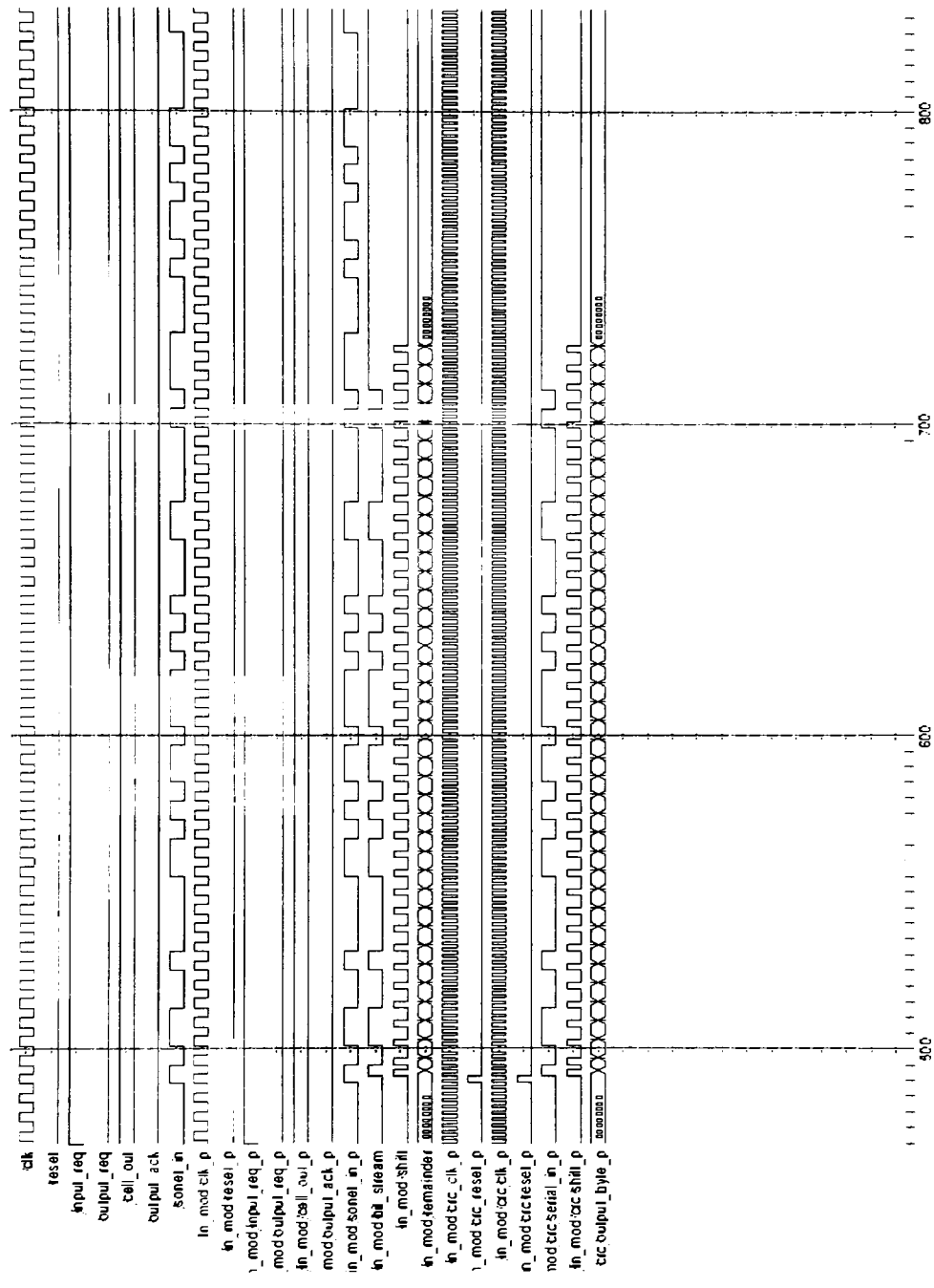
95 28 144 98 190 183 198 70 115 220
82 174 32 26 168 251 30 124 121 57
67 164 165 214 73 128 137 67 110 200
134 100 59 82 71 79 65 162 106 152
85 63 120 168 159 66 208 82 121 43
9 169 247

146 6 194 236 74 205 72 94 192 145
13 141 206 50 87 136 86 129 63 20
28 163 84 225 125 136 185 4 230 30
1 246 144 140 9 60 173 88 107 149
129 225 198 183 115 233 243 161 165 50
112 94 33

69 113 118 120 106 100 72 247 142 70
199 236 125 74 133 150 13 135 6 109
116 33 3 109 177 144 232 68 222 115
251 136 100 69 75 169 26 15 108 18
45 5 21 71 71 17 151 113 80 58
87 146 73

120 91 169 4 243 122 200 17 219 250
129 203 44 227 51 35 69 13 204 72
76 158 177 248 101 24 25 5 215 74
246 155 56 253 141 150 7 196 109 142
89 40 228 86 27 183 186 192 123 192
191 70 114

Waveform from simulation of input_module_tb.vhd



Appendix C

The following table contains a step by step example of the manner in which the header error control calculation is performed by the output module. The four header bytes for which the HEC value is being calculated are, in hexadecimal format, "01 02 03 04" Following these 32 bits, eight zeros are shifted in to form a complete 40-bit header. After all 40 bits have been shifted in, the register contains the hexadecimal value of "E3". This value is the correct HEC byte for this particular cell header. The output module will then insert his value into the fifth byte of the cell header before transmitting the cell over the SONET transport network.

	C-	C_6	C_5	C_4	C_3	C_2	C_1	C_0	$C_7 \oplus C_1$	$C_7 \oplus C_0$	$C_7 \oplus In$	In
Initial	0	0	0	0	0	0	0	0	0	0	0	0
Step 1	0	0	0	0	0	0	0	0	0	0	0	0
Step 2	0	0	0	0	0	0	0	0	0	0	0	0
Step 3	0	0	0	0	0	0	0	0	0	0	0	0
Step 4	0	0	0	0	0	0	0	0	0	0	0	0
Step 5	0	0	0	0	0	0	0	0	0	0	0	0
Step 6	0	0	0	0	0	0	0	0	0	0	0	0
Step 7	0	0	0	0	0	0	0	0	0	0	1	1
Step 8	0	0	0	0	0	0	0	1	0	1	0	0
Step 9	0	0	0	0	0	0	1	0	1	0	0	0
Step 10	0	0	0	0	0	1	0	0	0	0	0	0
Step 11	0	0	0	0	1	0	0	0	0	0	0	0
Step 12	0	0	0	1	0	0	0	0	0	0	0	0
Step 13	0	0	1	0	0	0	0	0	0	0	0	0
Step 14	0	1	0	0	0	0	0	0	0	0	1	1
Step 15	1	0	0	0	0	0	0	1	1	0	1	0
Step 16	0	0	0	0	0	1	0	1	0	1	0	0
Step 17	0	0	0	0	1	0	1	0	1	0	0	0
Step 18	0	0	0	1	0	1	0	0	0	0	0	0

Step 19	0	0	1	0	1	0	0	0	0	0	0	0
Step 20	0	1	0	1	0	0	0	0	0	0	0	0
Step 21	1	0	1	0	0	0	0	0	1	1	1	0
Step 22	0	1	0	0	0	1	1	1	1	1	1	1
Step 23	1	0	0	0	1	1	1	1	0	0	0	1
Step 24	0	0	0	1	1	0	0	0	0	0	0	0
Step 25	0	0	1	1	0	0	0	0	0	0	0	0
Step 26	0	1	1	0	0	0	0	0	0	0	0	0
Step 27	1	1	0	0	0	0	0	0	1	1	1	0
Step 28	1	0	0	0	0	1	1	1	0	0	1	0
Step 29	0	0	0	0	1	0	0	1	0	1	1	1
Step 30	0	0	0	1	0	0	1	1	1	1	0	0
Step 31	0	0	1	0	0	1	1	0	1	0	0	0
Step 32	0	1	0	0	1	1	0	0	0	0	0	0
Step 33	1	0	0	1	1	0	0	0	1	1	1	0
Step 34	0	0	1	1	0	1	1	1	1	1	0	0
Step 35	0	1	1	0	1	1	1	0	1	0	0	0
Step 36	1	1	0	1	1	1	0	0	1	1	1	0
Step 37	1	0	1	1	1	1	1	1	0	0	1	0
Step 38	0	1	1	1	1	0	0	1	0	1	0	0
Step 39	1	1	1	1	0	0	1	0	0	1	1	0
Step 40	1	1	1	0	0	0	1	1				

Appendix D

This appendix includes the results of a simple test case that was run on the output module. The "outmodcells.in" file shown here was used as input to both the C++ and the VHDL models. The file contains 40 ATM cells, out of which approximately twenty percent are idle. The 53 bytes of each cell are grouped together, and each cell is separated by a blank line. Each byte is represented in decimal format.

The C++ and the VHDL model each produced a file named "outmodframes.out". The contents of these files were identical and are included in this appendix. The file "outmodframes.out" contains the single SONET frame the was transmitted by the output module. Each one of the valid ATM cells from the file "outmodcells.in" has been mapped into this frame. The output module did not place any of the idle cells into the outgoing frame.

A sample waveform from the VHDL simulation of the entity "output_module_tb.vhd" is also included. This simulation used the file "outmodcells.in" as input. The time frame captured in this waveform shows the output module generating the HEC value for an outgoing ATM cell. The first four bytes of the cell's header are shifted into the "crc_check_e" one bit at a time. These 32 bits are followed by eight zero-bits. After all 40 bits have been shifted in, the correct HEC value for that cell's header is available.

59 99 249 210 194 169 90 194 223 156
37 238 227 98 181 12 48 11 223 70
242 89 237 220 242 139 86 10 200 199
211 57 176 112 121 162 101 208 16 176
43 111 63 197 128 110 209 60 11 202
45 200 77

0 0 0 0 0 210 5 91 133 0
105 241 93 185 240 4 201 99 249 91
109 67 252 145 35 182 13 33 135 50
95 53 217 77 63 65 186 86 91 207
213 41 235 147 254 74 245 162 216 117
197 113 2

118 151 201 233 56 252 176 115 169 227
46 244 214 146 171 123 227 61 148 240
231 46 138 197 211 98 195 56 69 156
235 176 3 170 5 223 144 91 38 237
128 99 24 225 125 38 152 9 167 32
223 26 54

177 76 153 1 169 38 218 12 79 72
114 247 80 107 102 115 126 149 175 6
98 151 153 123 4 141 251 80 3 134
120 172 44 135 75 253 101 224 113 13
43 158 196 175 124 129 60 111 117 202
121 194 234

108 128 232 152 156 80 134 163 115 44
55 250 201 194 161 106 151 111 74 155
220 130 167 175 180 184 178 230 193 240
4 168 213 228 16 156 59 102 188 43
212 88 241 125 250 92 222 213 67 244
19 107 31

0 0 0 0 0 121 176 187 24 144
123 253 67 155 92 225 50 199 229 176
87 108 181 101 101 228 105 253 127 91
144 163 254 193 213 59 17 107 8 202
127 146 158 203 122 56 130 60 18 159
45 20 83

0 0 0 0 0 162 91 84 61 243
64 1 188 116 151 217 76 161 0 69
210 87 196 153 148 16 160 22 62 69
29 32 40 31 28 217 102 240 84 232

42 205 203 153 248 147 165 162 223 74
197 188 9

167 180 184 176 34 204 134 236 225 216
132 4 54 203 82 209 229 249 154 91
77 192 210 78 69 187 87 172 251 175
168 28 208 251 225 246 60 245 159 8
212 8 119 231 120 111 73 10 173 243
95 101 61

226 104 136 199 153 245 49 132 7 60
73 7 174 164 13 72 0 210 181 239
199 171 225 131 117 230 14 195 185 26
53 23 250 88 167 149 18 124 234 39
254 194 164 182 246 202 236 112 124 158
121 141 241

158 156 88 222 113 32 220 156 171 32
140 10 167 251 72 64 153 44 80 5
66 149 239 183 38 18 69 219 120 132
193 19 36 54 237 52 231 2 181 69
169 252 81 132 117 166 143 214 74 73
19 54 38

0 0 0 0 0 73 7 53 208 132
81 13 33 212 3 56 52 5 107 154
188 127 254 109 214 61 251 242 54 110
78 15 204 146 179 82 188 7 1 100
84 55 126 210 244 129 51 61 25 243
45 222 218

0 0 0 0 0 242 177 204 117 104
149 16 154 173 190 48 78 94 6 47
55 233 13 161 7 105 178 137 244 216
217 138 245 112 248 240 18 140 76 130
254 241 170 160 115 220 86 163 230 30
197 135 15

0 0 0 0 0 29 220 228 154 204
90 19 20 5 248 166 231 55 33 69
177 211 155 86 183 21 233 161 178 67
102 134 32 204 190 143 231 18 151 161
168 44 87 238 242 184 249 10 180 200
95 48 195

217 208 39 245 68 70 135 125 63 48
158 22 141 221 179 158 2 144 187 217
44 190 169 139 231 64 160 184 112 45

242 130 73 170 132 46 189 23 226 191
83 103 132 188 113 148 156 112 131 115
121 216 247

21 133 246 141 34 111 178 22 100 21
99 25 7 182 238 150 155 105 213 238
166 40 184 64 151 107 215 79 47 151
126 125 241 7 202 76 147 156 46 222
253 33 49 138 240 239 64 214 81 30
19 129 44

207 185 198 165 73 153 93 173 9 120
167 157 128 14 169 14 181 194 113 132
33 19 198 117 72 150 142 103 236 2
138 121 27 228 144 234 105 161 121 252
168 91 93 216 238 202 226 61 31 200
172 42 225

0 0 0 0 0 194 8 197 46 92
108 160 249 230 100 5 80 27 12 153
156 252 213 169 120 193 69 126 171 108
23 245 69 65 86 137 190 39 68 28
82 150 10 166 109 38 7 163 237 115
197 210 22

0 0 0 0 0 236 51 94 210 192
175 163 115 62 159 252 105 244 39 46
23 231 227 94 41 109 125 21 105 86
162 240 237 31 156 167 147 172 143 59
252 80 55 245 236 2 169 10 187 30
95 249 202

11 236 149 188 35 22 221 246 247 164
116 166 236 23 90 244 4 77 193 68
18 81 242 147 89 153 52 45 40 192
47 236 23 123 97 69 105 177 218 89
167 138 227 195 107 221 77 112 138 72
248 162 254

70 161 101 211 79 63 9 14 156 9
184 169 101 239 149 108 29 39 219 216
140 60 1 72 10 196 234 68 229 43
187 232 64 88 39 227 63 56 38 120
82 197 17 145 234 57 239 215 88 242
19 75 179

1 213 53 234 217 105 179 166 193 108
125 172 222 71 80 99 183 127 119 237

7 38 16 125 186 239 34 91 163 21
72 227 233 181 109 130 148 189 113 150
251 0 188 223 105 21 20 62 38 157
172 243 231

0 0 0 0 0 146 222 63 102 81
193 175 88 32 11 91 209 89 145 131
129 17 30 50 233 27 216 242 98 127
211 96 19 146 51 160 106 194 188 181
166 186 233 173 231 239 182 164 244 72
69 156 28

0 0 0 0 0 188 137 86 139 180
134 178 209 248 70 210 107 177 45 152
251 122 45 102 154 198 143 10 32 233
96 92 60 112 248 63 64 72 8 211
81 244 23 251 103 75 90 11 194 241
95 69 208

61 10 133 130 160 102 52 238 48 152
202 181 75 80 1 202 6 139 71 46
118 101 186 155 202 241 198 33 222 84
236 87 228 204 63 221 22 77 210 242
124 47 195 201 229 39 252 113 144 156
248 237 5

120 62 84 154 216 143 95 135 85 252
143 184 196 41 59 193 32 227 226 67
240 79 201 80 123 30 125 184 156 62
120 83 14 170 5 124 234 210 30 18
38 233 239 151 101 130 160 215 95 71
146 150 186

51 241 36 177 112 185 10 32 249 224
210 187 62 128 246 57 185 189 125 215
107 185 215 133 44 73 52 207 90 168
5 207 56 7 201 154 64 88 105 48
208 36 156 230 227 94 195 62 173 241
172 63 238

110 38 243 200 242 226 53 55 31 69
151 190 183 89 177 49 211 22 151 237
230 163 229 58 92 244 107 230 25 19
145 202 97 228 16 57 22 93 180 79
123 95 201 180 226 58 103 164 124 28
69 231 163

0 0 0 0 0 13 223 207 195 41

92 193 49 50 236 41 109 239 51 130
97 142 244 110 13 32 34 126 214 252
29 198 10 65 213 215 235 226 0 237
38 25 118 3 97 148 10 11 74 198
222 16 215

0 0 0 0 0 54 138 104 232 141
160 196 170 137 167 32 135 72 77 23
219 120 3 36 60 75 216 149 22 103
169 194 51 31 28 245 193 104 76 13
207 83 162 208 223 112 173 113 24 113
248 184 12

169 90 194 223 46 95 181 128 13 240
101 199 35 98 225 151 34 34 232 45
86 226 18 88 236 118 16 172 211 209
54 189 93 123 225 147 151 109 151 43
122 142 79 31 95 76 80 215 230 28
146 97 192

100 15 146 119 195 137 96 24 178 212
169 202 156 59 157 143 59 122 3 193
208 204 32 141 29 161 198 196 145 187
194 58 6 88 167 50 235 243 98 74
37 200 124 236 221 167 116 62 180 198
44 10 244

159 67 98 142 195 178 139 176 214 57
110 205 22 146 88 135 213 84 157 215
75 183 47 66 205 78 254 91 80 38
78 54 47 181 237 208 193 248 173 104
207 130 41 186 93 131 23 164 130 113
69 178 169

0 0 0 0 0 219 54 73 124 29
178 208 15 107 146 254 238 172 184 108
197 161 61 118 126 121 181 114 14 144
218 49 88 146 178 238 151 126 248 135
121 189 85 9 219 222 186 12 81 28
222 91 221

0 0 0 0 0 6 96 97 160 129
118 211 136 194 78 245 137 133 83 129
64 12 76 44 174 164 108 138 204 250
103 45 2 239 120 141 109 4 68 166
36 247 2 214 91 185 94 114 31 70
120 4 147

218 118 50 166 227 47 12 248 69 101
186 214 2 155 136 237 163 222 238 23
186 245 217 96 95 207 163 33 138 228
242 41 43 204 190 44 67 9 143 196
206 178 47 38 217 149 1 89 237 240
146 172 199

150 43 1 189 162 217 182 145 106 201
127 217 123 116 68 101 61 183 9 44
53 224 232 22 143 250 90 56 73 79
127 164 84 42 132 202 152 142 218 227
121 236 92 243 88 240 37 191 187 155
44 85 124

209 95 208 212 32 3 225 169 15 45
195 220 244 203 254 93 215 17 163 192
175 74 246 74 63 166 17 80 7 184
12 160 125 7 74 232 110 147 38 2
35 39 9 193 215 204 199 38 137 70
196 125 176

13 147 160 235 109 44 140 65 52 17
136 223 110 164 57 84 240 233 190 214
43 53 6 254 239 210 72 103 196 162
151 156 39 100 144 135 67 25 240 33
205 225 53 16 213 168 107 140 88 239
222 37 228

0 0 0 0 0 86 182 217 216 117
204 226 231 125 244 76 139 67 89 107
165 31 20 52 32 253 254 253 131 13
163 151 80 65 86 38 25 158 60 63
120 28 225 221 85 4 14 242 38 154
120 205 153

199 72 239 131 31 127 98 241 253 89
145 229 97 212 47 195 165 28 116 129
32 10 35 104 208 41 181 22 65 119
48 147 121 158 27 68 238 163 135 94
163 86 15 44 211 222 177 89 244 69
18 118 205

0 0 0 0 0 0 0 0 0 0 59 99 249 210 194 169 90 194 223 156
37 238 227 98 181 12 48 11 223 70 242 89 237 220 242 139 86 10 200 199
211 57 176 112 121 162 101 208 16 176 43 111 63 197 128 110 209 60 11 202
45 200 77 118 151 201 233 56 252 176 115 169 227 46 244 214 146 171 123 227
61 148 240 231 46 138 197 211 98 195 56 69 156 235 176 3 170 5 223 144
91 38 237 128 99 24 225 125 38 152 9 167 32 223 26 54 177 76 153 1
169 38 218 12 79 72 114 247 80 107 102 115 126 149 175 6 98 151 153 123
4 141 251 80 3 134 120 172 44 135 75 253 101 224 113 13 43 158 196 175
124 129 60 111 117 202 121 194 234 108 128 232 152 156 80 134 163 115 44 55
250 201 194 161 106 151 111 74 155 220 130 167 175 180 184 178 230 193 240 4
168 213 228 16 156 59 102 188 43 212 88 241 125 250 92 222 213 67 244 19
107 31 167 180 184 176 34 204 134 236 225 216 132 4 54 203 82 209 229 249
154 91 77 192 210 78 69 187 87 172 251 175 168 28 208 251 225 246 60 245
159 8 212 8 119 231 120 111 73 10

0 0 0 0 0 0 0 0 0 0 173 243 95 101 61 226 104 136 199 153
245 49 132 7 60 73 7 174 164 13 72 0 210 181 239 199 171 225 131 117
230 14 195 185 26 53 23 250 88 167 149 18 124 234 39 254 194 164 182 246
202 236 112 124 158 121 141 241 158 156 88 222 113 32 220 156 171 32 140 10
167 251 72 64 153 44 80 5 66 149 239 183 38 18 69 219 120 132 193 19
36 54 237 52 231 2 181 69 169 252 81 132 117 166 143 214 74 73 19 54
38 217 208 39 245 68 70 135 125 63 48 158 22 141 221 179 158 2 144 187
217 44 190 169 139 231 64 160 184 112 45 242 130 73 170 132 46 189 23 226
191 83 103 132 188 113 148 156 112 131 115 121 216 247 21 133 246 141 34 111
178 22 100 21 99 25 7 182 238 150 155 105 213 238 166 40 184 64 151 107
215 79 47 151 126 125 241 7 202 76 147 156 46 222 253 33 49 138 240 239
64 214 81 30 19 129 44 207 185 198 165 73 153 93 173 9 120 167 157 128
14 169 14 181 194 113 132 33 19 198 117 72 150 142 103 236 2 138 121 27
228 144 234 105 161 121 252 168 91 93

0 0 0 0 0 0 0 0 0 0 216 238 202 226 61 31 200 172 42 225
11 236 149 188 35 22 221 246 247 164 116 166 236 23 90 244 4 77 193 68
18 81 242 147 89 153 52 45 40 192 47 236 23 123 97 69 105 177 218 89
167 138 227 195 107 221 77 112 138 72 248 162 254 70 161 101 211 79 63 9
14 156 9 184 169 101 239 149 108 29 39 219 216 140 60 1 72 10 196 234
68 229 43 187 232 64 88 39 227 63 56 38 120 82 197 17 145 234 57 239
215 88 242 19 75 179 1 213 53 234 217 105 179 166 193 108 125 172 222 71
80 99 183 127 119 237 7 38 16 125 186 239 34 91 163 21 72 227 233 181
109 130 148 189 113 150 251 0 188 223 105 21 20 62 38 157 172 243 231 61
10 133 130 160 102 52 238 48 152 202 181 75 80 1 202 6 139 71 46 118
101 186 155 202 241 198 33 222 84 236 87 228 204 63 221 22 77 210 242 124
47 195 201 229 39 252 113 144 156 248 237 5 120 62 84 154 216 143 95 135
85 252 143 184 196 41 59 193 32 227 226 67 240 79 201 80 123 30 125 184
156 62 120 83 14 170 5 124 234 210

0 0 0 0 0 0 0 0 0 0 30 18 38 233 239 151 101 130 160 215
95 71 146 150 186 51 241 36 177 112 185 10 32 249 224 210 187 62 128 246
57 185 189 125 215 107 185 215 133 44 73 52 207 90 168 5 207 56 7 201
154 64 88 105 48 208 36 156 230 227 94 195 62 173 241 172 63 238 110 38
243 200 242 226 53 55 31 69 151 190 183 89 177 49 211 22 151 237 230 163
229 58 92 244 107 230 25 19 145 202 97 228 16 57 22 93 180 79 123 95
201 180 226 58 103 164 124 28 69 231 163 169 90 194 223 46 95 181 128 13
240 101 199 35 98 225 151 34 34 232 45 86 226 18 88 236 118 16 172 211
209 54 189 93 123 225 147 151 109 151 43 122 142 79 31 95 76 80 215 230
28 146 97 192 100 15 146 119 195 137 96 24 178 212 169 202 156 59 157 143
59 122 3 193 208 204 32 141 29 161 198 196 145 187 194 58 6 88 167 50
235 243 98 74 37 200 124 236 221 167 116 62 180 198 44 10 244 159 67 98
142 195 178 139 176 214 57 110 205 22 146 88 135 213 84 157 215 75 183 47
66 205 78 254 91 80 38 78 54 47

0 0 0 0 0 0 0 0 0 0 181 237 208 193 248 173 104 207 130 41
186 93 131 23 164 130 113 69 178 169 218 118 50 166 227 47 12 248 69 101
186 214 2 155 136 237 163 222 238 23 186 245 217 96 95 207 163 33 138 228
242 41 43 204 190 44 67 9 143 196 206 178 47 38 217 149 1 89 237 240
146 172 199 150 43 1 189 162 217 182 145 106 201 127 217 123 116 68 101 61
183 9 44 53 224 232 22 143 250 90 56 73 79 127 164 84 42 132 202 152
142 218 227 121 236 92 243 88 240 37 191 187 155 44 85 124 209 95 208 212
32 3 225 169 15 45 195 220 244 203 254 93 215 17 163 192 175 74 246 74
63 166 17 80 7 184 12 160 125 7 74 232 110 147 38 2 35 39 9 193
215 204 199 38 137 70 196 125 176 13 147 160 235 109 44 140 65 52 17 136
223 110 164 57 84 240 233 190 214 43 53 6 254 239 210 72 103 196 162 151
156 39 100 144 135 67 25 240 33 205 225 53 16 213 168 107 140 88 239 222
37 228 199 72 239 131 31 127 98 241 253 89 145 229 97 212 47 195 165 28
116 129 32 10 35 104 208 41 181 22

0 0 0 0 0 0 0 0 0 0 65 119 48 147 121 158 27 68 238 163
135 94 163 86 15 44 211 222 177 89 244 69 18 118 205 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

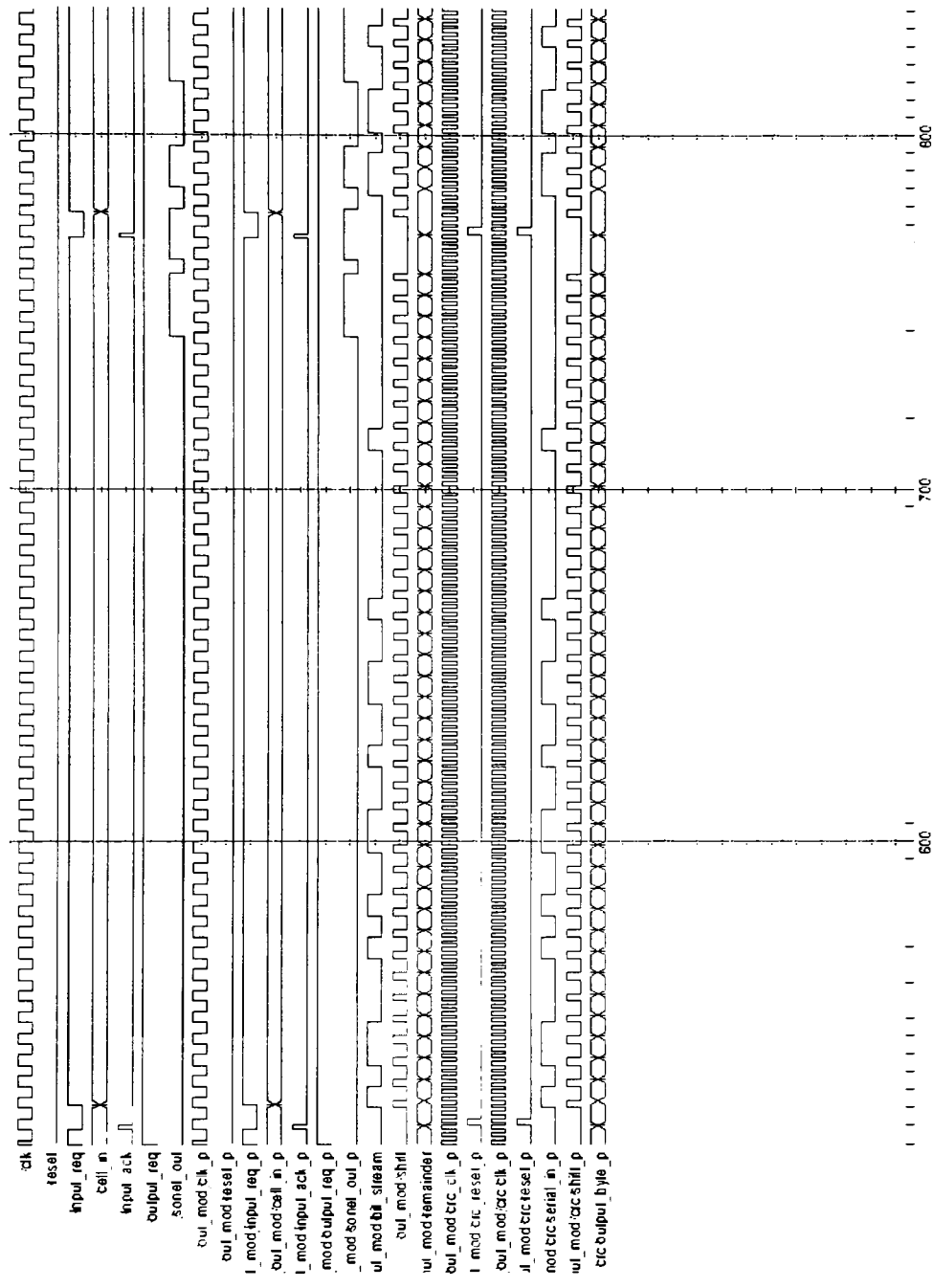
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0

Waveform from simulation of output_module_tb.vhd



Appendix E

This appendix provides a listing of each of the C++ source code files used to implement models of both an input module and an output module. The complete C++ code is contained within six files. They are "ATMCell.h", "ATMCell.cc", "SONETFrame.h", "SONETFrame.cc", "input.cc", and "output.cc". A "makefile" is also provided to simplify the task of compling these files.

```
SWITCH=-O3

input:    input.cc ATMCell.o SONETFrame.o
          g++ $(SWITCH) ATMCell.o SONETFrame.o input.cc -o input -lg++ -lm

output: output.cc output.h ATMCell.o SONETFrame.o
          g++ $(SWITCH) ATMCell.o SONETFrame.o output.cc -o output -lg++ -lm

ATMCell.o:  ATMCell.h ATMCell.cc
          g++ $(SWITCH) -c ATMCell.cc

SONETFrame.o:  SONETFrame.h SONETFrame.cc
          g++ $(SWITCH) -c SONETFrame.cc
```

ATMCell.h

```
/*************************************************************************
 NAME:     ATMCell.h
 AUTHOR:   Darin Murphy
 PURPOSE:  Defines an object that represents an ATM cell.
           This file contains all of the neccesary class
           declarations.
*************************************************************************/

#ifndef _ATMCELL_H
#define _ATMCELL_H

#define TRUE  1
#define FALSE 0
#define INVALIDCELLCHAR 'X'

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <time.h>

class ATMCell
{
  private:

  int isValid;

  unsigned header[5];
  unsigned payload[48];

  int syndromeTable[256];

  public:

  ATMCell();        //Constructor
  ~ATMCell();       //Destructor



/*************************************************************************
 NAME:     setCell
 PURPOSE:  Initializes the header and payload fields of the cell with
           the values passed in.
*************************************************************************/

  void ATMCell::setCell(int *newHeader, int *newPayload);
```

```
/*************************************************************************
 NAME:      validate
 PURPOSE:  Makes a cell valid
*************************************************************************/

  void ATMCell::validate();


/*************************************************************************
 NAME:      invalidate
 PURPOSE:  Makes a cell invalid
*************************************************************************/

  void ATMCell::invalidate();


/*************************************************************************
 NAME:      valid
 PURPOSE:  Returns TRUE if a cell is valid, returns FALSE if it is not
*************************************************************************/

  int ATMCell::valid();


/*************************************************************************
 NAME:      printCell
 PURPOSE: Prints out all 58 bytes of a cell
*************************************************************************/

  void ATMCell::printCell();


/*************************************************************************
 NAME:      generateHEC
 PURPOSE: Performs the necessary CRC calculations to generate
              the HEC byte
*************************************************************************/

  void ATMCell::generateHEC();


/*************************************************************************
 NAME:      validateHEC
 PURPOSE: Calculates the 8 bit CRC code for the first four bytes
             of the cell header and checks to make sure the existing
             HEC value in the 5th byte of the header matches the
             calculated value.  Returns TRUE if so, FALSE otherwise.
*************************************************************************/
```

```
  int ATMCell::validateHEC();


/*************************************************************************
 NAME:      genSynTable
 PURPOSE: Geneates the table of syndromes used to calculate
             the HEC byte
*************************************************************************/

  void ATMCell::genSynTable();


/*************************************************************************
 NAME:     randHeader
 PURPOSE: Inserts random bytes into all five bytes of the cell header
*************************************************************************/

  void ATMCell::randHeader();


/*************************************************************************
 NAME:      randPayload
 PURPOSE: Inserts random bytes into the entire payload of the ATM cell
*************************************************************************/

  void ATMCell::randPayload();


/*************************************************************************
 NAME:      returnContents
 PURPOSE: Returns all 53 bytes of data contained in the cell
*************************************************************************/

  void ATMCell::returnContents(int *data);


/*************************************************************************
 NAME:      writeToFile
 PURPOSE: Writes the contents of a cell to the specified file.
             The file must already be open for writing to.
*************************************************************************/

  void ATMCell::writeToFile(ofstream &tfile);
```

```
/*************************************************************************
 NAME:      sendCells
 PURPOSE: Generates the specified number of cells and writes thier
            contents to a file.
*************************************************************************/

  void ATMCell::sendCells();



/*************************************************************************
 NAME:      readInCell
 PURPOSE: This function reads in the contents of a cell from
            a file and places the data into the cell object
            passed to it.
*************************************************************************/

  void ATMCell::readInCell(ifstream& tfile);

};
#endif
```

```
/*************************************************************************
 NAME:     ATMCell.cc
 AUTHOR:   Darin Murphy
 PURPOSE:  Defines an object that represents an ATM cell.
           This file contains all of the neccesary class
           declarations.
*************************************************************************/

#include "ATMCell.h"

#define GENERATOR_POLY   0x107          /* x^8 + x^2 +  x  + 1  */
#define COSET_PATTERN    0x055          /* x^6 + x^4 + x^2 + 1  */

/*************************************************************************
 NAME:     ATMCell
 PURPOSE:  Constructor used to create an object of type ATMCell
*************************************************************************/

ATMCell::ATMCell()
{
  genSynTable();
  header[0] = 0x00;
  header[1] = 0x00;
  header[2] = 0x00;
  header[3] = 0x00;
  header[4] = 0x00;

  // All cells are initially valid

  isValid = TRUE;

}

/*************************************************************************
 NAME:    ~ATMCell
 PURPOSE:  Destructor that is called when an object of type ATMCell
           is destroyed
*************************************************************************/

ATMCell::~ATMCell()
{
}


/*************************************************************************
 NAME:     validate
 PURPOSE:  Makes a cell valid
*************************************************************************/
```

```
void ATMCell::validate()
{
  isValid = TRUE;
}


/*************************************************************************
 NAME:       invalidateCell
 PURPOSE:  Makes a cell valid
*************************************************************************/

void ATMCell::invalidate()
{
  isValid = FALSE;
  header[0] = 0x00;
  header[1] = 0x00;
  header[2] = 0x00;
  header[3] = 0x00;
  header[4] = 0x00;
  randPayload();
}


/*************************************************************************
 NAME:       valid
 PURPOSE:  Returns TRUE if a cell is valid, returns FALSE if it is not
*************************************************************************/

int ATMCell::valid()
{
  int   sum;
  int   i;

  sum = 0;

  for (i=0; i<4; i++)
  {
    sum = sum + header[i];
  }

  if (sum == 0)
  {
    return (FALSE);
  } else {
    return (TRUE);
  }
}


/*************************************************************************
 NAME:       setCell
 PURPOSE:  Initializes the header and payload fields of the cell with
           the values passed in.
*************************************************************************/
```

```
void ATMCell::setCell(int *newHeader, int *newPayload)
{
  int  i;

  for (i=0; i<5; i++)
  {
    header[i] = newHeader[i];
  }

  for (i=0; i<48; i++)
  {
    payload[i] = newPayload[i];
  }
}

/*************************************************************************
 NAME:      printCell
 PURPOSE:  Prints out all 58 bytes of a cell
*************************************************************************/

void ATMCell::printCell()
{
  int  i;

  cout << "\n\n";
  cout << "HEADER :\n\n" << header[0] << "  " << header[1]  << "  ";
  cout << header[2]  << "  " << header[3] << "  " << header[4] << "\n";
  cout << "\nPAYLOAD : " << "\n";
  for(i=0; i<48; i++)
  {
    if (i%8 == 0)
    {
      cout << "\n";
    }
    cout << payload[i] << "  ";
  }
  cout << "\n";
  cout << dec;
}


/*************************************************************************
 NAME:      generateHEC
 PURPOSE: Calculates the 8 bit CRC code for the first four bytes
              of the cell header and inserts the result into the HEC
              field.
*************************************************************************/

void ATMCell::generateHEC()
{
  int hec_accum = 0;
  int i;
```

```
  for ( i = 0;  i < 4;  i++ )
  {
    hec_accum = syndromeTable [ hec_accum ^ header[i] ];
  }
  header[4] = hec_accum ;

  return;
}


/*************************************************************************
 NAME:      validateHEC
 PURPOSE: Calculates the 8 bit CRC code for the first four bytes
              of the cell header and checks to make sure the existing
              HEC value in the 5th byte of the header matches the
              calculated value.  Returns TRUE if so, FALSE otherwise.
*************************************************************************/

int ATMCell::validateHEC()
{
  int hec_accum = 0;
  int i;

  for ( i = 0;  i < 4;  i++ )
  {
    hec_accum = syndromeTable [ hec_accum ^ header[i] ];
  }

  // Check if the current stored value is correct

  if (header[4] == (hec_accum))
  {
    return(TRUE);
  } else {
    if (header[0] != 0 && header[1] != 0 && header[2] != 0 && header[3] != 0)
    {
      for ( i = 0;  i < 4;  i++ )
      {
        cout << header[i] << "  ";
      }
    }

    return(FALSE);
  }

}


/*************************************************************************
 NAME:      genSynTable
 PURPOSE: Generate the table of syndromes used to calculate
              the HEC byte
*************************************************************************/
```

```
void ATMCell::genSynTable()
{
  int i;
  int j;
  int syndrome;
  for ( i = 0;  i < 256;  i++ )
  {
    syndrome = i;
    for ( j = 0;  j < 8;  j++ )
    {
      if ( syndrome & 0x80 )
        syndrome = ( syndrome << 1 ) ^ GENERATOR_POLY;
      else
        syndrome = ( syndrome << 1 );
    }
    syndromeTable[i] = syndrome;
  }
  return;
}

/*************************************************************************
 NAME:       randHeader
 PURPOSE:  Insert random bytes into all five bytes of the cell header
*************************************************************************/

void ATMCell::randHeader()
{
  int       i;
  static int  factor;

  factor++;
  srand( ((unsigned) time((time_t*)NULL)) * factor );
  for (i=0; i<5; i++)
  {
    header[i] = rand()%255;
  }
}

/*************************************************************************
 NAME:       randPayload
 PURPOSE:  Insert random bytes into the entire payload section of the
           ATM cell
*************************************************************************/

void ATMCell::randPayload()
{
  int i;
  static int  factor;
  factor++;
  srand(((unsigned) time((time_t*)NULL) ) * (factor*18));
  for (i=0; i<48; i++){
    payload[i] = rand()%255;
  }
}
```

```
/*************************************************************************
 NAME:      returnContents
 PURPOSE: Returns all 53 bytes of data contained in the cell
*************************************************************************/

void ATMCell::returnContents(int *data)
{
  int  i;

  for (i=0; i<5; i++)
  {
    data[i] = header[i];
  }

  for (i=0; i<48; i++)
  {
    data[i+5] = payload[i];
  }
}


/*************************************************************************
 NAME:      writeToFile
 PURPOSE: Writes the contents of a cell to the specified file.
             The file must already be open for writing to.
*************************************************************************/

void ATMCell::writeToFile(ofstream &tfile)
{
  int  i;
  int  j;

  for (i=0; i<5; i++)
  {
    if (i != 0)
    {
      tfile << " ";
    }
    tfile << header[i];
  }
  for (j=0; j<48; j++)
  {
    if ((j+5)%10 == 0)
    {
      tfile << "\n";
      tfile << payload[j];
    } else {
      tfile << " " << payload[j];
    }
  }
  tfile << "\n";
  tfile << "\n";
}
```

```
/*************************************************************************
 NAME:     sendCells
 PURPOSE: Generates the specified number of cells and writes their
          contents to a file.
*************************************************************************/

void ATMCell::sendCells()
{
  int         numCells;
  int         i;
  int         percentValid;
  ATMCell     tempCell;
  ofstream    outfile;


  cout << "\nInput the number of ATM cells to be generated : ";
  cin >> numCells;
  cout << "\n\n";

  cout << "Input the percentage of these cells that should be valid : ";
  cin >> percentValid;
  cout << "\n\n";

  outfile.open("outmodcells.in");

  // Seed the RNG only once

  srand( (unsigned) time((time_t*)NULL));

  for (i=0; i<numCells; i++)
  {

    // Based on the given percentage, each cell is
    // given a random chance of being valid

    if ( rand()% 100 <= percentValid )
    {
      tempCell.validate();
      tempCell.randHeader();
      tempCell.randPayload();
      tempCell.generateHEC();
    } else {
      tempCell.invalidate();
    }

    tempCell.writeToFile(outfile);

  }

  outfile.close();

}
```

```
/*************************************************************************
 NAME:      readInCell
 PURPOSE:  This function reads in the contents of a cell from
           a file and places the data into the cell object
           passed to it.
*************************************************************************/
void ATMCell::readInCell(ifstream& tfile)
{
  int        i;
  int        j;
  char       ch;
  char       charArray[4];
  int        intArray[53];
  static int numCellsRead = 0;


  numCellsRead++;
  this.validate();

  for (i=0; i<53; i++)
  {

    // Check for a premature EOF

    if ( tfile.eof() )
    {
          cout << "\nAn incomplete cell has been found in the input file!\n";
    }

    // Remove whitespace between bytes

    while (tfile.peek() == ' ' || tfile.peek() == '\n')
    {
      tfile.get(ch);
    }

    if (tfile.peek() == INVALIDCELLCHAR)
    {
          this.invalidate();
      tfile.get(ch);

    } else {

      // Read in the next byte

      j = 0;
      while (tfile.peek() != ' ' && tfile.peek() != '\n' &&
              tfile.peek() != EOF)
      {
        tfile.get(ch);
        charArray[j] = ch;
            j++;
```

```
      }
      charArray[j] = '\0';
      intArray[i] = atoi(charArray);
    }
  }

  this.setCell(intArray, intArray+5);

  // Remove any trailing whitespace between cells

  while (tfile.peek() == ' ' || tfile.peek() == '\n')
  {
    tfile.get(ch);
  }
}
```

SONETFrame.h

```
/*************************************************************************
 NAME:     SONETFrame.h
 AUTHOR:   Darin Murphy
 PURPOSE:  Defines an object that represents a SONET frame.
           This file contains all of the neccesary class
           declarations.
*************************************************************************/

#ifndef _SONETFRAME_H
#define _SONETFRAME_H

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <time.h>

class SONETFrame
{
  private:

  unsigned  sts3c[9][270];

  public:

  SONETFrame();         //Constructor
  ~SONETFrame();        //Destructor



/*************************************************************************
 NAME:     clear
 PURPOSE:  Clears the contents of the frame to all zeros.
*************************************************************************/

  void SONETFrame::clear();



/*************************************************************************
 NAME:     setPayload
 PURPOSE:  Sets the payload field of the sts3c SONET frame with
           the values stored in the array passed to it.  The overhead
           bytes of the frame are not affected.
*************************************************************************/

  void SONETFrame::setPayload(int *data);
```

```
/*************************************************************************
 NAME:      returnSPE
 PURPOSE:  Returns the payload field of the sts3c SONET frame by
           storing the values in the array passed to it.  The overhead
           bytes of the frame are not reutrned, only the actual data.
*************************************************************************/

  void SONETFrame::returnSPE(int *data);



/*************************************************************************
 NAME:      writeToFile
 PURPOSE:  Write the contents of a complete frame to a file.
*************************************************************************/

  void SONETFrame::writeToFile(ofstream &tfile);



/*************************************************************************
 NAME:      readInFrame
 PURPOSE:  Reads the next frame from the file that is passed to it,
           and stores the contents of the frame into the current
           object.  The file must already be open for reading.
*************************************************************************/

  void SONETFrame::readInFrame(ifstream &tfile);


};
#endif
```

```
/*************************************************************************
 NAME:     SONETFrame.cc
 AUTHOR:   Darin Murphy
 PURPOSE:  Defines an object that represents a SONET frame.
           This file contains all of the neccesary class
           declarations.
*************************************************************************/

#include "SONETFrame.h"
#include <iostream.h>
#include <stdlib.h>
#include <time.h>


/*************************************************************************
 NAME:     SONETFrame
 PURPOSE:  Constructor used to create an object of type SONETFrame
*************************************************************************/

SONETFrame::SONETFrame()
{
  int i;
  int j;

  for (i=0; i<9; i++)
  {
    for (j=0; j<270; j++)
    {
          sts3c[i][j] = 0;
    }
  }
}


/*************************************************************************
 NAME:     ~SONETFrame
 PURPOSE:  Destructor that is called when an object of type SONETFrame
           is destroyed
*************************************************************************/
SONETFrame::~SONETFrame()
{
}


/*************************************************************************
 NAME:     clear
 PURPOSE:  Clears the contents of the frame to all zeros.
*************************************************************************/
```

```
void SONETFrame::clear()
{
  int i;
  int j;

  for (i=0; i<9; i++)
  {
    for (j=0; j<270; j++)
    {
          sts3c[i][j] = 0;
    }
  }
}


/*************************************************************************
 NAME:      setPayload
 PURPOSE:  Sets the payload field of the sts3c SONET frame with
           the values stored in the array passed to it.  The overhead
           bytes of the frame are not affected.
*************************************************************************/

void SONETFrame::setPayload(int *newPayload)
{
  int i;
  int j;
  int k;

  k=0;
  for (i=0; i<9; i++)
  {
    for (j=10; j<270; j++)
    {
          sts3c[i][j] = newPayload[k];
          k++;
    }
  }
}


/*************************************************************************
 NAME:      returnSPE
 PURPOSE:  Returns the payload field of the sts3c SONET frame by
           storing the values in the array passed to it.  The overhead
           bytes of the frame are not reutrned, only the actual data.
*************************************************************************/

void SONETFrame::returnSPE(int *data)
{
  int i;
  int j;
  int k;

  k=0;
```

```
  for (i=0; i<9; i++)
  {
    for (j=10; j<270; j++)
    {
          data[k] = sts3c[i][j];
          k++;
    }
  }
}


/*************************************************************************
 NAME:      writeToFile
 PURPOSE:  Write the contents of a complete frame to a file.
*************************************************************************/

void SONETFrame::writeToFile(ofstream &tfile)
{
  int  i;
  int  j;

  for (i=0; i<9; i++)
  {
    for (j=0; j<270; j++)
    {
          if (j%20 == 0)
          {
        tfile << "\n";
            tfile << sts3c[i][j];
          } else {
            tfile << " " << sts3c[i][j];
          }

    }
    tfile << "\n";
  }

  tfile << "\n\n\n\n";

}


/*************************************************************************
 NAME:      readInFrame
 PURPOSE:  Reads the next frame from the file that is passed to it,
            and stores the contents of the frame into the current
            object.  The file must already be open for reading.
*************************************************************************/

void SONETFrame::readInFrame(ifstream &tfile)
{
  int       i;
  int       j;
  int       k;
```

```
    char        ch;
    char        charArray[4];
    static int  numFramesRead = 0;

    numFramesRead++;

    for (i=0; i<9; i++)
    {
      for (j=0; j<270; j++)
      {

        // Check for a premature EOF

        if ( tfile.eof() )
        {
          cout << "\nAn incomplete frame has been found in the input file!\n";
        }

        // Remove whitespace between bytes

        while (tfile.peek() == ' ' || tfile.peek() == '\n')
        {
          tfile.get(ch);
        }

        // Read in the next byte

        k = 0;
        while (tfile.peek() != ' ' && tfile.peek() != '\n' &&
            tfile.peek() != EOF)
        {
          tfile.get(ch);
          charArray[k] = ch;
          k++;
        }
        charArray[k] = '\0';
        sts3c[i][j] = atoi(charArray);
      }
    }

    // Remove any trailing whitspace between frames

    while (tfile.peek() == ' ' || tfile.peek() == '\n')
    {
      tfile.get(ch);
    }
}
```

```
/*************************************************************************
 NAME:     input.cc
 AUTHOR:   Darin Murphy
 PURPOSE:  Verifys the functionality of an input module for
           and ATM switch.
*************************************************************************/

#include <iostream.h>
#include "ATMCell.h"
#include "SONETFrame.h"


main()
{

  SONETFrame  tempFrame;
  ATMCell     tempCell;
  ifstream    infile;
  ofstream    outfile;
  int         spe[2340]:
  int         speIndex;
  int         bytesMissing:
  int         headerSum;
  int         overflow;
  int         ovrflwArray[53];
  int         ovrflwIndex:
  int         i;


  infile.open("inmodframes.in");
  outfile.open("inmodcells.out"):
  bytesMissing = 0;
  overflow = FALSE;
  speIndex = 0;

  while (!infile.eof())
  {

    // Read in the next frame from the input file

    tempFrame.readInFrame(infile):
    tempFrame.returnSPE(spe);
    speIndex = 0;

    if (overflow)
    {
        for (i=0; i<bytesMissing; i++)
        {
```

```
        ovrflwArray[ovrflwIndex] = spe[speIndex];
        ovrflwIndex++;
        speIndex++;
  }

      tempCell.setCell( (ovrflwArray), (ovrflwArray+5) );

  // Check the HEC byte

      if (tempCell.validateHEC())
      {
    if (tempCell.valid())
    {
          tempCell.writeToFile(outfile);
    }
      } else {
        cout << "An invalid HEC byte has been found in the input\n";
      }

      overflow = FALSE;

}

while (speIndex + 53 < 2340)
{
      tempCell.setCell( (spe+speIndex), (spe+speIndex+5) );

  // Check the HEC byte

      if (tempCell.validateHEC())
      {
    if (tempCell.valid())
    {
          tempCell.writeToFile(outfile);
    }
      } else {

        headerSum = 0;

        for (i=0; i<4; i++)
        {
          headerSum = headerSum + *(spe+speIndex+i);
        }

        if (headerSum != 0)
        {
          cout << "An invalid HEC byte has been found in the input\n";
        }
      }

      speIndex = speIndex + 53;
}

if (speIndex != 2340)
```

```
    {
      speIndex = speIndex + 53;
          overflow = TRUE;
          ovrflwIndex = 0;
      bytesMissing = speIndex - 2340;
          speIndex = speIndex - 53;
          while (speIndex < 2340)
          {
            ovrflwArray[ovrflwIndex] = spe[speIndex];
            ovrflwIndex++;
            speIndex++;
      }
    }
  }

  infile.close();
  outfile.close();

}
```

```
/*************************************************************************
 NAME:     output.cc
 AUTHOR:   Darin Murphy
 PURPOSE:  Verifys the functionality of an output module for
           an ATM switch.
*************************************************************************/

#include <iostream.h>
#include <fstream.h>
#include "ATMCell.h"
#include "SONETFrame.h"

#define  TRUE  1
#define  FALSE 0

main()
{
  ATMCell       tempCell;
  SONETFrame    tempFrame;
  ifstream      infile;
  ofstream      outfile;
  int           spe[2340];
  int           cellData[53];
  int           bytesInSPE;
  int           index;
  int           overflow;
  int           i;
  char          yesorno;


  cout << "\nDo you wish to generate a new input file containing a \n";
  cout << "specified number of random ATM cells? (y/n): ";
  cin >> yesorno;

  if (yesorno == 'y')
  {
    tempCell.sendCells();
  }

  bytesInSPE = 0;
  overflow = FALSE;
  infile.open("outmodcells.in");
  outfile.open("outmodframes.out");

  while ( (!infile.eof()) || (overflow) )
  {

    // Get the next ATM cell
```

```
  if (!overflow)
  {
    tempCell.readInCell(infile);
    tempCell.returnContents(cellData);
    index = 0;
  }

  // Map the current ATM cell to the proper location in the SPE
  // provided that the cell is a valid one

  if (tempCell.valid())
  {
    while ((bytesInSPE < 2340) && (index < 53))
    {
          spe[bytesInSPE] = cellData[index];
          bytesInSPE++;
          index++;
    }
  } else {
        index = 53;
  }

  // Write to the frame once the payload has been filled

  if (bytesInSPE == 2340)
  {
        tempFrame.setPayload(spe);
        tempFrame.writeToFile(outfile);
        tempFrame.clear();
    bytesInSPE = 0;

        // Clear the SPE

        for (i=0; i<2340; i++)
        {
          spe[i] = 0;
        }

  }

  // Check to see if a cell has passed the frame boundary

  if (index < 53) {
        overflow = TRUE;
  } else {
        overflow = FALSE;
  }

}

// If the last frame is only partially used, write it to the file

if (bytesInSPE != 0)
```

```
    {
      tempFrame.setPayload(spe);
      tempFrame.writeToFile(outfile);
      tempFrame.clear();
      bytesInSPE = 0;
    }

    infile.close();
    outfile.close();

}
```

Appendix F

This appendix provides all of the VHDL source code files that are used to implement models of both an input module and an output module. The complete VHDL code is contained within eight separate files. These are "inoutmod_pd.vhd", "xor_ea.vhd", "shift_ea.vhd", "crc_check_ea.vhd", "input_module_ea.vhd", "input_module_tb.vhd", "output_module_ea.vhd", and "output_module_tb.vhd." Each of these files is listed in the following pages.

```
------------------------------------------------------------------
-- NAME:     Input/Output Module Package
-- AUTHOR:  Darin Murphy
-- PURPOSE: Defines all several types, functions, and constants
--              that are used for generating a VHDL model of
--              both an input and output module that will interface
--              a SONET network to an ATM switch.
------------------------------------------------------------------

------------------------------------------------------------------
-- Package declaration
------------------------------------------------------------------

PACKAGE inoutmod_Pkg IS

  ------------------------------------------------------------------
  -- Subtypes
  ------------------------------------------------------------------

  SUBTYPE cell_Typ  IS BIT_VECTOR (0 TO 423);  --53 bytes in a cell
  SUBTYPE byte_Typ  IS BIT_VECTOR (0 TO 7);    --8 bits in a byte

  -- The 9 bytes of path overhead are not included in the following
  -- definition of an STS-3c SPE.  They are included in the definition
  -- of a complete frame (frame_Typ)

  SUBTYPE spe_Typ   IS BIT_VECTOR (0 TO 18719); --2,340 bytes in sts3c spe
  SUBTYPE frame_Typ IS BIT_VECTOR (0 TO 19439); --2,430 bytes in frame

  ------------------------------------------------------------------
  -- Functions
  ------------------------------------------------------------------


  ------------------------------------------------------------------
  -- NAME:     chartobyte
  -- PURPOSE:  Converts a character to a byte
  ------------------------------------------------------------------

  FUNCTION chartobyte (char : CHARACTER) RETURN BIT_VECTOR;

  ------------------------------------------------------------------
  -- NAME:     inttobitvector
  -- PURPOSE:  Converts an integer to a bit vector
  --              The first argument is the integer to be converted.
  --              The second argument is the length of the bit vector
  --              to be returned.
  ------------------------------------------------------------------
```

```
  FUNCTION inttobitvector (data : INTEGER;
                           size : INTEGER) RETURN BIT_VECTOR;


  ------------------------------------------------------------------
  -- NAME:      bytetoint
  -- PURPOSE:   Converts an array of 8 bits into the corresponding
  --            integer value. The argument must be a bit vector
  --            of 8 bits containing the byte to be converted.
  ------------------------------------------------------------------

  FUNCTION bytetoint (byte : BIT_VECTOR) RETURN INTEGER;


  ------------------------------------------------------------------
  -- NAME:      bytetochar
  -- PURPOSE:   Converts an array of 8 bits into the corresponding
  --            character value. The argument must be of type
  --            byte_Typ and must contain the byte to be converted.
  ------------------------------------------------------------------

  FUNCTION bytetochar (byte : byte_Typ) RETURN INTEGER;


END inoutmod_Pkg;


------------------------------------------------------------------
-- Package body
------------------------------------------------------------------

PACKAGE BODY inoutmod_Pkg IS

  ------------------------------------------------------------------
  -- NAME:      chartobyte
  -- PURPOSE:   Converts a character to a byte
  ------------------------------------------------------------------

  FUNCTION chartobyte (char : CHARACTER) RETURN BIT_VECTOR IS

    VARIABLE byte      : BIT_VECTOR(0 TO 7);  -- Byte to be returned
    VARIABLE ordinal   : INTEGER := 0;        -- ASCII value of character
    VARIABLE remainder : INTEGER := 0;        -- Remainder when divided by 2

  BEGIN

    -- Convert the character to an integer value

    ordinal := CHARACTER'POS(char);

    -- Generate the bit pattern that represents that integer using
    -- repeated division by 2. The remainder indicates the value of
    -- the next bit.
```

```
  FOR index IN byte'RANGE LOOP

        remainder := ordinal rem 2;
        ordinal := ordinal / 2;

        IF remainder = 0 THEN
          byte(byte'HIGH - index) := '0';
    ELSE
          byte(byte'HIGH - index) := '1';
    END IF;

  END LOOP;

  RETURN byte;

END chartobyte;


-------------------------------------------------------------------
-- NAME:       inttobitvector
-- PURPOSE:  Converts an integer to a bit vector.
--                The first argument is the integer to be converted.
--                The second argument is the length of the bit vector
--                 to be returned.
-------------------------------------------------------------------

FUNCTION inttobitvector (data : INTEGER;
                                size : INTEGER) RETURN BIT_VECTOR IS

  VARIABLE vector      : BIT_VECTOR (0 TO size - 1);
  VARIABLE remainder  : INTEGER := 0;
  VARIABLE new_data   : INTEGER := 0;

BEGIN

  new_data := data;

  -- Perform conversion by repeatedly dividing the integer value
  -- by 2 and using the remainder to determine the next bit

  FOR index IN vector'RANGE LOOP

        remainder := new_data rem 2;
        new_data := new_data / 2;

        IF remainder = 0 THEN
          vector(vector'HIGH - index) := '0';
    ELSE
          vector(vector'HIGH - index) := '1';
    END IF;

  END LOOP;
```

```
  RETURN vector;

END inttobitvector;


-------------------------------------------------------------------
-- NAME:       bytetoint
-- PURPOSE:  Converts an array of 8 bits into the corresponding
--                integer value.  The argument must be a bit vector
--                of 8 bits containing the byte to be converted.
-------------------------------------------------------------------

FUNCTION bytetoint (byte : BIT_VECTOR) RETURN INTEGER IS

  VARIABLE int_value   : INTEGER := 0;

BEGIN

  FOR index IN byte'RANGE LOOP

        IF byte(index) = '1' THEN
          int_value := int_value + (2 ** index);
    END IF;

  END LOOP;

  RETURN int_value;

END bytetoint;


-------------------------------------------------------------------
-- NAME:       bytetochar
-- PURPOSE:  Converts an array of 8 bits into the corresponding
--                character value.  The argument must be of type
--                byte_Typ and must contain the byte to be converted.
-------------------------------------------------------------------

FUNCTION bytetochar (byte : byte_Typ) RETURN INTEGER IS

  VARIABLE temp_sum : INTEGER := 0;  -- stores the temporary sum

BEGIN

  FOR index IN byte'RANGE LOOP

        IF byte(index) = '1' THEN

          temp_sum := temp_sum + (2 ** (abs(index-7)));

    END IF;

  END LOOP;
```

```
        RETURN (temp_sum);
    END bytetochar;
END inoutmod_Pkg;
```

```
-------------------------------------------------------------------
-- NAME:     XOR Gate Entity/Architecture
-- AUTHOR:  Darin Murphy
-- PURPOSE: Model of a two input XOR gate
-------------------------------------------------------------------


-------------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------------

ENTITY xor2_e IS
  PORT (x_p  :  IN  BIT;
        y_p  :  IN  BIT;
        z_p  :  OUT BIT);
END xor2_e;


-------------------------------------------------------------------
-- Architecture declaration
-------------------------------------------------------------------

ARCHITECTURE behavior_a OF xor2_e IS

BEGIN

  z_p <= x_p XOR y_p AFTER 1 ps;

END behavior_a;
```

```
-------------------------------------------------------------------
-- NAME:     Shift Register Entity/Architecture
-- AUTHOR:  Darin Murphy
-- PURPOSE: Model of a one bit shift register
-------------------------------------------------------------------

LIBRARY ARITHMETIC;
USE ARITHMETIC.std_logic_arith.ALL;

ENTITY shift_e IS

  PORT (clk_p      :  IN  BIT;
        clear_p    :  IN  BIT;
        shift_p    :  IN  BIT;
        in_bit_p   :  IN  BIT;
        out_bit_p  :  OUT BIT);

END shift_e;

ARCHITECTURE behavior_a OF shift_e IS

  CONSTANT shift_c     : TIME := 0 NS;
  CONSTANT clear_c     : TIME := 0 NS;
  CONSTANT load_c      : TIME := 0 NS;

  SIGNAL   next_output_s : BIT;

BEGIN

   -- This process models the behavior of a 1-bit shift register

   sreg : PROCESS(clk_p, clear_p)

   BEGIN

         -- Clear the register if requested

     IF (clear_p = '1') THEN
          next_output_s <= '0' AFTER clear_c;
     ELSE

          -- Shift the input through to the output

       IF ((clk_p'EVENT) AND (clk_p = '1') AND (shift_p = '1')) THEN

         next_output_s <= in_bit_p AFTER shift_c;

       ELSE
```

```
                next_output_s <= next_output_s;

        END IF;

    END IF;

  END PROCESS sreg;


  -- This process updates the contents of the register each time
  -- it changes

  update : PROCESS(next_output_s)

  BEGIN

        out_bit_p <= next_output_s;

  END PROCESS update;

END behavior_a;
```

```
-------------------------------------------------------------------
-- NAME:      Entity/Architecture for the CRC checking hardware
-- AUTHOR:    Darin Murphy
-- PURPOSE:   Model of the hardware used to perform an 8-bit
--            CRC check of an ATM cell header using the generator
--            polynomial X^8 + X^2 + X + 1
-------------------------------------------------------------------

LIBRARY work;
USE work.inoutmod_Pkg.ALL;

-------------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------------

ENTITY crc_check_e IS

  PORT (clk_p          : IN BIT;
        reset_p        : IN BIT;
        serial_in_p    : IN BIT;
        shift_p        : IN BIT;
        output_byte_p : INOUT byte_Typ);

END crc_check_e;


-------------------------------------------------------------------
-- Architecture declaration
-------------------------------------------------------------------

ARCHITECTURE structural_a OF crc_check_e IS

  SIGNAL x1_out_s  : BIT;
  SIGNAL x2_out_s  : BIT;
  SIGNAL x3_out_s  : BIT;

  COMPONENT xor2_e
    PORT (x_p : IN BIT;
          y_p : IN BIT;
          z_p : OUT BIT);
  END COMPONENT;


  COMPONENT shift_e
    PORT (clk_p     : IN  BIT;
          clear_p   : IN  BIT;
          shift_p   : IN  BIT;
          in_bit_p  : IN  BIT;
```

```
      out_bit_p   : OUT BIT);
  END COMPONENT;

BEGIN

  X1 : xor2_e    port map (x_p => serial_in_p, y_p => output_byte_p(7),
                           z_p => x1_out_s);

  C0 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => x1_out_s,
                         out_bit_p => output_byte_p(0));

  X2 : xor2_e    port map (x_p => output_byte_p(0), y_p => output_byte_p(7),
                           z_p => x2_out_s);

  C1 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => x2_out_s,
                         out_bit_p => output_byte_p(1));

  X3 : xor2_e    port map (x_p => output_byte_p(1), y_p => output_byte_p(7),
                           z_p => x3_out_s);

  C2 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => x3_out_s,
                         out_bit_p => output_byte_p(2));

  C3 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => output_byte_p(2),
                         out_bit_p => output_byte_p(3));

  C4 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => output_byte_p(3),
                         out_bit_p => output_byte_p(4));

  C5 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => output_byte_p(4),
                         out_bit_p => output_byte_p(5));

  C6 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => output_byte_p(5),
                         out_bit_p => output_byte_p(6));

  C7 : shift_e port map (clk_p => clk_p, clear_p => reset_p,
                         shift_p => shift_p,
                         in_bit_p => output_byte_p(6),
                         out_bit_p => output_byte_p(7));
END structural_a;
```

```
-------------------------------------------------------------------
-- NAME:     Input Module Entity/Architecture
-- AUTHOR:  Darin Murphy
-- PURPOSE: Model of an input module that receives ATM cells
--               as input and maps the cells into the payload of
--               a SONET frame.
-------------------------------------------------------------------

LIBRARY work;
USE work.inoutmod_Pkg.ALL;

-------------------------------------------------------------------
-- Entity declaration
-------------------------------------------------------------------

ENTITY input_module_e IS

  PORT (clk_p          : IN BIT;
        input_req_p    : IN BIT := '0';
        output_req_p   : OUT BIT := '0';
        cell_out_p     : OUT cell_Typ;
        output_ack_p   : IN BIT := '0';
        sonet_in_p     : IN BIT);

END input_module_e;


-------------------------------------------------------------------
-- Architecture declaration
-------------------------------------------------------------------

ARCHITECTURE input_a OF input_module_e IS

  SIGNAL   bit_stream_s  : BIT := '0';
  SIGNAL   shift_s       : BIT := '0';
  SIGNAL   remainder_s   : byte_Typ;
  SIGNAL   crc_clk_s     : BIT := '0';
  SIGNAL   crc_reset_s   : BIT := '0';

  COMPONENT crc_check_e
    PORT (clk_p       : IN BIT;
          reset_p     : IN BIT;
          serial_in_p : IN BIT;
          shift_p     : IN BIT;
          output_byte_p : INOUT byte_Typ);
  END COMPONENT;

BEGIN
```

```
-- Component instantiation of crc_check_e entity

crc : crc_check_e  port map (clk_p => crc_clk_s, reset_p => crc_reset_s,
                   serial_in_p => bit_stream_s,
                   shift_p => shift_s,
                   output_byte_p => remainder_s);

-- Process to generate the clock signal to drive the crc_check_e entity
-- This clock has a period of 2 ns

crc_clk_gen : PROCESS(crc_clk_s)

BEGIN

  crc_clk_s <= NOT crc_clk_s AFTER 1 ns;

END PROCESS crc_clk_gen;


-- The main process used to simulate the behavior of the input module

main : PROCESS

VARIABLE cell_one         : cell_Typ;
VARIABLE cell_two         : cell_Typ;
VARIABLE header_one       : BIT_VECTOR(0 TO 39);
VARIABLE cell_one_index   : INTEGER := 0;
VARIABLE cell_two_index   : INTEGER := 0;
VARIABLE bits_shifted     : INTEGER := 0;
VARIABLE bits_input       : INTEGER := 0;
VARIABLE checking_hec     : BOOLEAN := FALSE;
VARIABLE clr_crc_chk      : BOOLEAN := TRUE;
VARIABLE hunt             : BOOLEAN := TRUE;
VARIABLE presynch         : BOOLEAN := FALSE;
VARIABLE synch            : BOOLEAN := FALSE;
VARIABLE idle_cell        : BOOLEAN := FALSE;

BEGIN

  WAIT UNTIL ((clk_p'EVENT) AND (clk_p = '1') AND (input_req_p = '1'));

  -- Ignore SONET overhead bits

  IF (bits_input mod (270*8) >= 0) AND (bits_input mod (270*8) <= 79) THEN

    bits_input := bits_input + 1;

  ELSE

    -- Identify the current state of the cell delineation process

    IF (hunt) THEN
```

```
IF (clr_crc_chk) THEN

  -- Reset the crc_check entity to contain all zeros
  -- each time a new header is being processed

  crc_reset_s <= '1';
  WAIT FOR 2 ns;
  crc_reset_s <= '0';

  clr_crc_chk := FALSE;
  header_one := (OTHERS => '0');

END IF;

IF (remainder_s = "00000000") AND (bits_shifted > 39) THEN

  hunt := FALSE;
  presynch := TRUE;
  synch := FALSE;
  cell_one_index := 41;

  -- Copy the valid header into the proper location of the first
  -- cell. The presynch state will fill in the rest of this cell

  cell_one(0 TO 39) := header_one(0 TO 39);
  cell_one(40) := sonet_in_p;
  bits_input := bits_input + 1;


  IF (header_one = "0000000000000000000000000000000000000000")
    THEN

    idle_cell := TRUE;

  END IF;

ELSIF (input_req_p = '1') THEN

  -- Shift the next incoming bit into the crc checking hardware

  shift_s <= '1';
  bit_stream_s <= sonet_in_p;
  WAIT FOR 2 ns;
  shift_s <= '0';
  bits_shifted := bits_shifted + 1;
  bits_input := bits_input + 1;

  -- Shift the current header to the left by one bit
  -- MSB(0) . . . LSB(39)

  FOR i IN 0 TO 38 LOOP
    header_one(i) := header_one(i+1);
  END LOOP;
```

```
    -- Store the next incoming bit into temporary register

    header_one(39) := sonet_in_p;

  END IF;

ELSIF (presynch) OR (synch) THEN

  IF (cell_one_index <= 423) THEN

    -- Shift in the cell payload data

    cell_one(cell_one_index) := sonet_in_p;
    cell_one_index := cell_one_index + 1;
    bits_input := bits_input + 1;

  ELSIF (cell_one_index = 424) THEN

    IF NOT (idle_cell) THEN

      -- Output the last complete cell to be read in

      cell_out_p <= cell_one;
      output_req_p <= '1';
      WAIT FOR 1 ns;
      output_req_p <= '0';

    END IF;

    idle_cell := FALSE;

    header_one := (OTHERS => '0');
    bits_shifted := 0;
    checking_hec := TRUE;
    cell_one_index := 999;

  END IF;

  IF (checking_hec) THEN

    IF (remainder_s = "00000000") AND (bits_shifted > 39) THEN

      -- Another valid cell has been encountered
      -- Cell boundaries have been correctly established

      hunt := FALSE;
      presynch := FALSE;
      synch := TRUE;

      checking_hec := FALSE;
      cell_one_index := 41;

      -- Copy valid header into the proper location of the second
      -- cell.  The synch state will fill in the rest of this cell
```

```
            cell_one(0 TO 39) := header_one(0 TO 39);
            cell_one(40) := sonet_in_p;
            bits_input := bits_input + 1;

            IF (header_one = "0000000000000000000000000000000000000000")
              THEN

              idle_cell := TRUE;

            END IF;

          ELSIF (input_req_p = '1') THEN

            -- Shift the next 40 bits into the crc checking hardware

            shift_s <= '1';
            bit_stream_s <= sonet_in_p;
            WAIT FOR 2 ns;
            shift_s <= '0';
            bits_shifted := bits_shifted + 1;
            bits_input := bits_input + 1;

            -- Shift the current header to the left by one bit
            -- MSB(0)  . . LSB(39)

            FOR i IN 0 TO 38 LOOP
              header_one(i) := header_one(i+1);
            END LOOP;

            -- Store the next incoming bit into temporary register

            header_one(39) := sonet_in_p;

          END IF;

        END IF;

      ELSIF (synch) THEN

      END IF;

    END IF;

  END PROCESS main;

END input_a;
```

```
------------------------------------------------------------------
-- NAME:      Input Module Testbench
-- AUTHOR:  Darin Murphy
-- PURPOSE: Testbench to test to proper behavior of the input
--                module entity.
------------------------------------------------------------------

LIBRARY std;
USE std.textio.ALL;
LIBRARY work;
USE work.inoutmod_Pkg.ALL;

------------------------------------------------------------------
-- Entity declaration
------------------------------------------------------------------

ENTITY input_module_tb IS

END input_module_tb;


------------------------------------------------------------------
-- Architecture declaration
------------------------------------------------------------------

ARCHITECTURE testbench_a OF input_module_tb IS

  SIGNAL  clk_s        : BIT;
  SIGNAL  input_req_s   : BIT;
  SIGNAL  output_req_s  : BIT;
  SIGNAL  cell_out_s    : cell_Typ;
  SIGNAL  output_ack_s  : BIT;
  SIGNAL  sonet_in_s    : BIT;

  COMPONENT input_module_e
    PORT (clk_p        : IN BIT;
          input_req_p   : IN BIT := '0';
          output_req_p  : OUT BIT := '0';
          cell_out_p    : OUT cell_Typ;
          output_ack_p  : IN BIT := '0';
          sonet_in_p    : IN BIT);
  END COMPONENT;

BEGIN

  -- Component instantiation of the input module to be teseted

  in_mod : input_module_e  port map(clk_p => clk_s,
```

```
                    input_req_p => input_req_s,
                    output_req_p => output_req_s,
                    cell_out_p => cell_out_s,
                                output_ack_p => output_ack_s,
                    sonet_in_p => sonet_in_s);


-- Generate a 6.43 ns period clock to drive the input module being tested

clock : PROCESS

BEGIN

  clk_s <= '0';
  WAIT FOR 3.215 ns;
  clk_s <= '1';
  WAIT FOR 3.215 ns;

END PROCESS clock;


-- Process to send SONET frames to the input module
-- In a complete switch the input module would receive frames
-- from the SONET transport network
-- Input is read from an ASCII file

input_side : PROCESS

  VARIABLE line_buffer  : LINE;
  VARIABLE byte_count   : INTEGER := 0;
  VARIABLE frame        : frame_Typ;
  VARIABLE frame_index  : INTEGER := 0;
  VARIABLE next_intbyte : INTEGER;
  VARIABLE out_frame_full : BOOLEAN := FALSE;

  FILE    input_file   : text IS IN "inmodframes.in";

BEGIN

WAIT UNTIL (clk_s'EVENT) AND (clk_s = '1');

  IF (out_frame_full) THEN

    -- Send the next bit of the output frame

    -- Delay until next clock cycle
    WAIT FOR 6 ns;

    sonet_in_s <= frame(frame_index);
    frame_index := frame_index + 1;

    IF (frame_index > frame_Typ'HIGH) THEN

      frame_index := 0;
```

```
    out_frame_full := FALSE;

  END IF;

ELSE

  -- Read in SONET frames from the ASCII input file

  IF ((NOT ENDFILE(input_file))) THEN

    -- Read the next line from the input file into a buffer

    READLINE(input_file, line_buffer);

    WHILE (line_buffer'LENGTH = 0) AND (NOT ENDFILE(input_file)) LOOP
      READLINE(input_file. line_buffer);
    END LOOP;

    -- Extract data from current line

    WHILE (byte_count < 2430) AND (line_buffer'LENGTH /= 0) LOOP

      READ(line_buffer, next_intbyte);

      frame((byte_count*8) TO ((byte_count*8) + 7)) :=
          inttobitvector(next_intbyte, 8);

      byte_count := byte_count + 1;

      -- Read in the next line when necessary

      IF (byte_count < 2430) THEN

        WHILE (line_buffer'LENGTH = 0) AND
            (NOT ENDFILE(input_file)) LOOP

          READLINE(input_file, line_buffer);

        END LOOP;

      END IF;

    END LOOP;

    -- Check to see if an entire frame has been read in

    IF (byte_count = 2430) THEN

      -- Output the first bit of the new frame

      -- Delay until next clock cycle
      WAIT FOR 6 ns;

      input_req_s <= '1';
```

```
        sonet_in_s <= frame(frame_index);
        frame_index := frame_index + 1;

            -- Prepare to read in the next cell from the ASCII file

        out_frame_full := TRUE;
        byte_count := 0;

      END IF;

    END IF;

  END IF;

END PROCESS input_side;


-- Process to receive ATM cells from the input module
-- In a complete switch the input module would pass cells to
-- the cell switch fabric
-- All output is written to an ASCII file

output_side : PROCESS

  VARIABLE line_buffer  : LINE;
  VARIABLE cell         : cell_Typ;
  VARIABLE cell_index   : INTEGER := 0;
  VARIABLE bytes_in_line: INTEGER := 0;
  VARIABLE bytes_written: INTEGER := 0;
  VARIABLE next_intbyte : INTEGER;
  VARIABLE byte_start   : INTEGER := 0;
  VARIABLE temp_byte    : byte_Typ;

  FILE    output_file  : text IS OUT "inmodcells.out";

BEGIN

  WAIT UNTIL output_req_s = '1';

  cell := cell_out_s;
  output_ack_s <= '1';
  WAIT FOR 1 ns;
  output_ack_s <= '0';

  -- When a complete cell has been read, write it to an output file

  bytes_in_line := 0;
  bytes_written := 0;

  FOR index IN 0 TO 52 LOOP  -- 53 bytes in a complete cell

    -- Convert each byte to a decimal value

    byte_start := index*8;
```

```
        temp_byte := cell( (byte_start) TO ( (byte_start) + 7) );
        next_intbyte := bytetochar(temp_byte);
        bytes_written := bytes_written + 1;

        IF (bytes_in_line < 10) THEN

          -- Output the next byte on the current line

          WRITE(line_buffer, next_intbyte);

          -- Don't add a trailing space

          IF (bytes_in_line /= 9) AND (index /= 52) THEN
            WRITE(line_buffer, ' ');
          END IF;

          bytes_in_line := bytes_in_line + 1;

        ELSIF (bytes_in_line = 10) THEN

          -- Flush the output buffer and begin a new line

          WRITELINE (output_file, line_buffer);
          WRITE(line_buffer, next_intbyte);
          WRITE(line_buffer, ' ');
          bytes_in_line := 1;

        END IF;

      END LOOP;

      -- Insert whitespace to separate cells within the output file

      WRITE(line_buffer, LF);
      WRITELINE (output_file, line_buffer);

    END PROCESS output_side;

END testbench_a;
```

```
--------------------------------------------------------------------
-- NAME:     Output Module Entity/Architecture
-- AUTHOR:  Darin Murphy
-- PURPOSE: Model of an output module that receives ATM cells
--               as input and maps the cells into the payload of
--               a SONET frame.
--------------------------------------------------------------------

LIBRARY work;
USE work.inoutmod_Pkg.ALL;

--------------------------------------------------------------------
-- Entity declaration
--------------------------------------------------------------------

ENTITY output_module_e IS

  PORT (clk_p        : IN BIT;
        input_req_p   : IN BIT := '0';
        cell_in_p     : IN cell_Typ;
        input_ack_p   : OUT BIT := '0';
        output_req_p  : OUT BIT := '0';
        sonet_out_p   : OUT BIT);

END output_module_e;


--------------------------------------------------------------------
-- Architecture declaration
--------------------------------------------------------------------

ARCHITECTURE output_a OF output_module_e IS

  SIGNAL   bit_stream_s  : BIT := '0';
  SIGNAL   shift_s       : BIT := '0';
  SIGNAL   remainder_s   : byte_Typ;
  SIGNAL   crc_clk_s     : BIT := '0';
  SIGNAL   crc_reset_s   : BIT := '0';

  COMPONENT crc_check_e
    PORT (clk_p        : IN BIT;
          reset_p      : IN BIT;
          serial_in_p  : IN BIT;
          shift_p      : IN BIT;
          output_byte_p : INOUT byte_Typ);
  END COMPONENT;

BEGIN
```

```
-- Component instantiation of crc_check_e entity

crc : crc_check_e  port map (clk_p => crc_clk_s, reset_p => crc_reset_s,
                   serial_in_p => bit_stream_s,
                   shift_p => shift_s,
                   output_byte_p => remainder_s);

-- Process to generate the clock signal to drive the crc_check_e entity
-- This clock has a period of 2 ns

crc_clk_gen : PROCESS(crc_clk_s)

BEGIN

  crc_clk_s <= NOT crc_clk_s AFTER 1 ns;

END PROCESS crc_clk_gen;



-- The main process used to simulate the behavior of the output module

main : PROCESS

VARIABLE new_cell       : cell_Typ;
VARIABLE over_flow_buff  : cell_Typ;
VARIABLE cell_index      : INTEGER := 0;
VARIABLE over_flow_index  : INTEGER := 0;
VARIABLE curr_frame      : frame_Typ;
VARIABLE frame_index     : INTEGER := 0;
VARIABLE output_index    : INTEGER := 0;
VARIABLE next_frame      : frame_Typ;
VARIABLE bits_shifted    : INTEGER := 0;
VARIABLE clr_crc_chk     : BOOLEAN := TRUE;
VARIABLE send_next_bit   : BOOLEAN := FALSE;
VARIABLE curr_frame_full  : BOOLEAN := FALSE;
VARIABLE next_frame_full  : BOOLEAN := FALSE;
VARIABLE new_cell_valid   : BOOLEAN := FALSE;
VARIABLE last_frame      : BOOLEAN := FALSE;
VARIABLE num_zero_bits   : INTEGER := 0;

BEGIN

  WAIT UNTIL ((clk_p'EVENT) AND (clk_p = '1'));

  IF (NOT next_frame_full) THEN

    -- Generate the HEC byte for the next outgoing cell

    IF (clr_crc_chk) AND (input_req_p = '1') THEN

      -- Read in and acknowledge the next ATM cell
```

```
  new_cell := cell_in_p;

  -- Ignore idle cells

  IF (new_cell(0 TO 39) /=
    "0000000000000000000000000000000000000000") THEN

    new_cell_valid := TRUE;
    input_ack_p <= '1';
    WAIT FOR 1 ns;
    input_ack_p <= '0';

    -- Reset the crc_check entity to contain all zeros
    -- each time a new cell is processed

    crc_reset_s <= '1';
    WAIT FOR 2 ns;
    crc_reset_s <= '0';
    clr_crc_chk := FALSE;
  ELSE
    input_ack_p <= '1';
    WAIT FOR 1 ns;
    input_ack_p <= '0';
  END IF;

ELSIF (bits_shifted < 32) AND (new_cell_valid) THEN

  -- Shift in the next bit of the current cell's header

  shift_s <= '1';
  bit_stream_s <= new_cell(bits_shifted);
  WAIT FOR 2 ns;
  shift_s <= '0';
  bits_shifted := bits_shifted + 1;

ELSIF (bits_shifted = 32) AND (num_zero_bits = 8) THEN

  -- Place the HEC byte that was just calculated into the
      -- fifth byte of the cell header

  new_cell(32) := remainder_s(7);
  new_cell(33) := remainder_s(6);
  new_cell(34) := remainder_s(5);
  new_cell(35) := remainder_s(4);
  new_cell(36) := remainder_s(3);
  new_cell(37) := remainder_s(2);
  new_cell(38) := remainder_s(1);
  new_cell(39) := remainder_s(0);

  -- Map the new cell into the appropriate SONET frame

  -- First insert any portion of a cell that overflowed from
  -- the last frame
```

```
IF (over_flow_index > 0) THEN

  -- The first 10 bytes of each row are SONET overhead
  -- Set each of these bits to 0

  FOR i IN 0 TO (10*8-1) LOOP
    next_frame(frame_index) := '0';
    frame_index := frame_index + 1;
  END LOOP;

  FOR index IN 0 TO (over_flow_index - 1) LOOP

    next_frame(frame_index) := over_flow_buff(index);
    frame_index := frame_index + 1;

  END LOOP;

  over_flow_index := 0;

END IF;

-- Map the next complete cell into the proper frame

cell_index := 0;

WHILE ((cell_index <= cell_Typ'HIGH) AND
     (frame_index <= frame_Typ'HIGH)) LOOP

  -- Insert SONET overhead bytes

  IF (frame_index MOD (270*8) = 0) THEN

    -- The first 10 bytes of each row are SONET overhead
    -- Set each of these bits to 0

    FOR i IN 0 TO (10*8-1) LOOP

      IF (curr_frame_full) THEN
        next_frame(frame_index) := '0';
      ELSE
        curr_frame(frame_index) := '0';
      END IF;

      frame_index := frame_index + 1;

    END LOOP;

  END IF;

  IF (curr_frame_full) THEN
    next_frame(frame_index) := new_cell(cell_index);
  ELSE
    curr_frame(frame_index) := new_cell(cell_index);
  END IF;
```

```
    frame_index := frame_index + 1;
    cell_index := cell_index + 1;

  END LOOP;

  -- Check to see if an entire frame has been filled
  -- If so, place the rest of 'new_cell' into a temporary
  -- buffer so that it can be inserted when the next frame
  -- is generated

  IF (frame_index > frame_Typ'HIGH) THEN

    IF (curr_frame_full) THEN
      next_frame_full := TRUE;
    ELSE
      curr_frame_full := TRUE;
    END IF;

    over_flow_index := 0;

    WHILE (cell_index <= cell_Typ'HIGH) LOOP

      over_flow_buff(over_flow_index) := new_cell(cell_index);
      over_flow_index := over_flow_index + 1;
      cell_index := cell_index + 1;

    END LOOP;

    frame_index := 0;

  END IF;

  -- Indicate that the first SONET frame is ready for transmission

      output_req_p <= '1';
      send_next_bit := TRUE;

  -- Prepare to calculate HEC byte of the next cell

  clr_crc_chk := TRUE;
  new_cell_valid := FALSE;
      bits_shifted := 0;
      num_zero_bits := 0;

ELSIF (bits_shifted = 32) THEN

  -- Shift in 8 zeros after all four header bytes

  shift_s <= '1';
  bit_stream_s <= '0';
  WAIT FOR 2 ns;
  shift_s <= '0';
  num_zero_bits := num_zero_bits + 1;
```

```
    END IF;

  END IF;

  IF (send_next_bit) THEN

    sonet_out_p <= curr_frame(output_index);

    IF (output_index = curr_frame'HIGH) AND (next_frame_full) THEN

      -- Copy next_frame into curr_frame

      frame_index := 0;

      WHILE (frame_index <= curr_frame'HIGH) LOOP
        curr_frame(frame_index) := next_frame(frame_index);
        frame_index := frame_index + 1;
      END LOOP;

      output_index := 0;
      frame_index := 0;
      next_frame_full := FALSE;

    ELSIF (output_index = curr_frame'HIGH) AND NOT (last_frame) THEN

      -- Fill the next frame with zeros and copy it into the current
      -- frame to be output

      WHILE (frame_index <= frame_Typ'HIGH) LOOP
        next_frame(frame_index) := '0';
        frame_index := frame_index + 1;
      END LOOP;

      frame_index := 0;

      WHILE (frame_index <= curr_frame'HIGH) LOOP
        curr_frame(frame_index) := next_frame(frame_index);
        frame_index := frame_index + 1;
      END LOOP;

      last_frame := TRUE;
      output_index := 0;
      frame_index := 0;
      next_frame_full := FALSE;

    ELSIF (output_index = curr_frame'HIGH) AND (last_frame) THEN

      send_next_bit := FALSE;
      output_req_p <= '0' AFTER 100 ns;

    ELSE

      output_index := output_index + 1;
```

```
        END IF;
      END IF;
    END PROCESS main;
END output_a;
```

```
------------------------------------------------------------------
-- NAME:     Output Module Testbench
-- AUTHOR:  Darin Murphy
-- PURPOSE: Testbench to test to proper behavior of the output
--                module entity.
------------------------------------------------------------------

LIBRARY std;
USE std.textio.ALL;
LIBRARY work;
USE work.inoutmod_Pkg.ALL;

------------------------------------------------------------------
-- Entity declaration
------------------------------------------------------------------

ENTITY output_module_tb IS

END output_module_tb;


------------------------------------------------------------------
-- Architecture declaration
------------------------------------------------------------------

ARCHITECTURE testbench_a OF output_module_tb IS

  SIGNAL  clk_s        : BIT;
  SIGNAL  input_req_s   : BIT;
  SIGNAL  cell_in_s     : cell_Typ;
  SIGNAL  input_ack_s   : BIT;
  SIGNAL  output_req_s  : BIT;
  SIGNAL  sonet_out_s   : BIT;

  COMPONENT output_module_e
    PORT (clk_p         : IN BIT;
          input_req_p   : IN BIT := '0';
          cell_in_p     : IN cell_Typ;
          input_ack_p   : OUT BIT := '0';
          output_req_p  : OUT BIT := '0';
          sonet_out_p   : OUT BIT);
  END COMPONENT;

BEGIN

  -- Component instantiation of the output module to be teseted

  out_mod : output_module_e  port map(clk_p => clk_s,
```

```
                    input_req_p => input_req_s,
                    cell_in_p => cell_in_s,
                                input_ack_p => input_ack_s,
                    output_req_p => output_req_s,
                    sonet_out_p => sonet_out_s);


-- Generate a 6.43 ns period clock to drive the output module being tested

clock : PROCESS

BEGIN

  clk_s <= '0';
  WAIT FOR 3.215 ns;
  clk_s <= '1';
  WAIT FOR 3.215 ns;

END PROCESS clock;


-- Process to send ATM cells to the output module
-- In a complete switch the output module would receive cells
-- from the switch fabric
-- Input is read from an ASCII file

input_side : PROCESS

  VARIABLE line_buffer  : LINE;
  VARIABLE byte_count   : INTEGER := 0;
  VARIABLE cell         : cell_Typ;
  VARIABLE cell_index   : INTEGER := 0;
  VARIABLE next_intbyte : INTEGER;
  VARIABLE byte_start   : INTEGER := 0;

  FILE    input_file   : text IS IN "outmodcells.in";

BEGIN

  -- Read in ATM cells from the ASCII input file

  IF ((NOT ENDFILE(input_file))) THEN

        -- Read the next line from the input file into a buffer

        READLINE(input_file, line_buffer);

        -- Extract cell data from current line

        WHILE ((byte_count < 53) AND (line_buffer'LENGTH > 0)) LOOP

      READ(line_buffer, next_intbyte);
      cell((byte_count*8) TO ((byte_count*8) + 7)) :=
          inttobitvector(next_intbyte, 8);
```

```
        byte_count := byte_count + 1;

      END LOOP;

      -- Check to see if an entire cell has been read in

         IF (byte_count = 53) THEN

        -- Place the next cell at the input port of the output module

        cell_in_s <= cell;

        -- Indicate to the output module that a new cell is present

        input_req_s <= '1';

        -- Wait until the output module has read that cell

        WAIT UNTIL input_ack_s = '1';

            -- Prepare to read in the next cell from the ASCII file

        input_req_s <= '0';
        WAIT UNTIL ((clk_s'EVENT) AND (clk_s = '1'));
        WAIT FOR 1 ns;
            byte_count := 0;

      END IF;

    ELSE

      -- Once the entire input file has been read, suspend this process

      input_req_s <= '0';
      WAIT;

    END IF;

  END PROCESS input_side;


  -- Process to receive ATM cells from the output module
  -- In a complete switch the output module would pass cells to
  -- the SONET transport network
  -- All output is written to an ASCII file

  output_side : PROCESS(clk_s)

    VARIABLE frame       : frame_Typ;
    VARIABLE frame_index  : INTEGER := 0;
    VARIABLE line_buffer  : LINE;
    VARIABLE bytes_in_line: INTEGER := 0;
    VARIABLE bytes_written: INTEGER := 0;
    VARIABLE next_intbyte : INTEGER;
```

```
  VARIABLE byte_start   : INTEGER := 0;
  VARIABLE temp_byte    : byte_Typ;

  FILE     output_file  : text IS OUT "outmodframes.out";

BEGIN

  IF ((clk_s'EVENT) AND (clk_s = '1') AND (output_req_s = '1')) THEN

    -- Read in the next bit of the frame that is being transmitted
    -- by the output module

    frame(frame_index) := sonet_out_s;
    frame_index := frame_index + 1;

    -- Once a complete frame has been filled, write it to an output
    -- file

    IF (frame_index > frame'HIGH) THEN

      bytes_in_line := 0;
      bytes_written := 0;

      FOR index IN 0 TO 2429 LOOP  -- 2430 bytes in a complete frame

        -- Convert each byte to a decimal value

        byte_start := index*8;
        temp_byte := frame( (byte_start) TO ( (byte_start) + 7) );
        next_intbyte := bytetochar(temp_byte);
        bytes_written := bytes_written + 1;

        IF ((bytes_in_line < 20) AND (bytes_written < 270)) THEN

          -- Output the next byte on the current line

          WRITE(line_buffer, next_intbyte);

          -- Don't add a trailing space

          IF (bytes_in_line /= 19) THEN
            WRITE(line_buffer, ' ');
          END IF;

          bytes_in_line := bytes_in_line + 1;

        ELSIF ((bytes_in_line = 20) AND (bytes_written < 270)) THEN

          -- Flush the output buffer and begin a new line

          WRITELINE (output_file, line_buffer);
          WRITE(line_buffer, next_intbyte);
          WRITE(line_buffer, ' ');
          bytes_in_line := 1;
```

```
      ELSIF (bytes_written = 270) THEN

        -- Skip to the next row of the current frame

        WRITE(line_buffer, next_intbyte);
        WRITE(line_buffer, LF);
        WRITELINE (output_file, line_buffer);
        bytes_written := 0;
        bytes_in_line := 0;

      END IF;

    END LOOP;

    -- Insert whitespace to separate frames within the output file

    WRITE(line_buffer, LF);
    WRITE(line_buffer, LF);
    WRITE(line_buffer, LF);
    WRITELINE (output_file, line_buffer);

    -- Prepare to receive next frame

    frame_index := 0;

  END IF;

END IF;
```

8 444BR 5831
11/97 TH
04-172-00 GBC